Beginner's Best Guide to Programming

スラスラわかる JavaScript

アシアル株式会社
生形 可奈子 著

特長

知識ゼロ
HTML・CSSの知識は不要

徹底理解
理解を助ける図解・サンプル

基礎＋α
プログラム自作の知識とコツ



▶サンプルプログラムは公式Webページからダウンロードできます

SE SHOEISHA

プログラミングの世界へようこそ！

初めてプログラミングを学ぶ人を対象にした"スラスラわかる"シリーズの新刊です。本書では、「JavaScript関連の知識はゼロ」という状態から、JavaScriptの基礎知識や作法、基本構文、基本テクニックなどを無理なく習得し、かつプログラムが自力で作れるようになるところまで、しっかりとフォローします。JavaScriptの基礎固めをしっかりしたい方だけでなく、ほかの入門書で挫折した方やプログラミングに苦手意識を持っている方にもおすすめの一冊です。

＊　＊　＊　＊　＊　＊　＊　＊　＊　＊　＊　＊　＊　＊

本書がテーマとしている「JavaScript」は、Internet ExplorerやGoogle Chromeなどに代表されるブラウザで利用されるプログラミング言語であり、いまやWebアプリケーション開発に欠かせない知識です。

本書ではJavaScriptプログラミングの基礎を無理なく学べるよう、図解と短いサンプルを使用したわかりやすい解説を目指しました。JavaScriptプログラミングの第一歩として、ぜひ本書をお役立てください。

スラスラわかるシリーズのWebページ

http://www.shoeisha.co.jp/book/surasura/

・サンプルプログラムのソースコード　・補足資料　・追加情報・・・　など

Beginner's Best Guide to Programming

スラスラわかる JavaScript

アシアル株式会社
生形 可奈子 著



SE
SHOEISHA

翔泳社ecoProjectのご案内

株式会社 翔泳社では地球にやさしい本づくりを目指します。
制作工程において以下の基準を定め、このうち4項目以上を満たしたものをエコロジー製品と位置づけ、シンボルマークをつけています。

資材	基準	期待される効果	本書採用
装丁用紙	無塩素漂白パルプ使用紙 あるいは 再生循環資源を利用した紙	有毒な有機塩素化合物発生の軽減（無塩素漂白パルプ） 資源の再生循環促進（再生循環資源紙）	○
本文用紙	材料の一部に無塩素漂白パルプ あるいは 古紙を利用	有毒な有機塩素化合物発生の軽減（無塩素漂白パルプ） ごみ減量・資源の有効活用（再生紙）	○
製版	CTP（フィルムを介さずデータから直接プレートを作製する方法）	枯渇資源（原油）の保護、産業廃棄物排出量の減少	○
印刷インキ*	植物油を含んだインキ	枯渇資源（原油）の保護、生産可能な農業資源の有効利用	○
製本メルト	難細裂化ホットメルト	細裂化しないために再生紙生産時に不純物としての回収が容易	
装丁加工	植物性樹脂フィルムを使用した加工 あるいは フィルム無使用加工	枯渇資源（原油）の保護、生産可能な農業資源の有効利用	

*：パール、メタリック、蛍光インキを除く

本書内容に関するお問い合わせについて

本書に関するご質問、正誤表については、下記のWebサイトをご参照ください。

正誤表　http://www.shoeisha.co.jp/book/errata/
刊行物Q&A　http://www.shoeisha.co.jp/book/qa/

インターネットをご利用でない場合は、FAXまたは郵便で、下記にお問い合わせください。

〒160-0006　東京都新宿区舟町5
（株）翔泳社 愛読者サービスセンター
FAX番号：03-5362-3818

電話でのご質問は、お受けしておりません。

※本書に記載されたURL等は予告なく変更される場合があります。
※本書の出版にあたっては正確な記述につとめましたが、著者や出版社などのいずれも、本書の内容に対してなんらかの保証をするものではなく、内容やサンプルに基づくいかなる運用結果に関してもいっさいの責任を負いません。
※本書に掲載されているサンプルプログラムやスクリプト、および実行結果を記した画面イメージなどは、特定の設定に基づいた環境にて再現される一例です。

はじめに

ようこそJavaScriptの世界へ！

本書を手に取っていただき、どうもありがとうございます。

この本に興味を持ってくださったということは、きっと自分の力でWebサイトを作ることを目的としている方なのでしょう。

では、あなたの作りたいWebサイトは、どのようなものでしょうか？

かゆいところに手が届く、ユーザーフレンドリーなWebサイトですか？　それとも、縦横無尽に動くアニメーションがちりばめられた、イマドキのWebサイトでしょうか？　あるいは、誰も作ったことのない、あなただけの独創性にあふれたWebサイトでしょうか？

本書は、そんなあなたの手助けをするためのツールです。あなたの目標を実現するために、JavaScriptの基本を習得するまでの間、学習の道標となるのが本書の役割です。

もしかしたら、学習を進めていくうちに「JavaScriptを書くのは難しい」とつまづくこともあるかもしれません。そんなときは、最初に本書を手に取ったときの目的を思い出してみてください。大切なのは、自らの歩みを止めないことです。少しずつでも着実に進むことで、あなたが頭に思い描いたWebサイトを、必ず現実のものにすることができます。

本書を読んでいただいた方々の制作したWebサイトが、インターネット上にあふれかえる日を楽しみにしています。

アシアル株式会社
生形 可奈子

本書について

本書は、Webサイトがどのように作られているのかを知らない、プログラミングもしたことがない、という初心者の方に向けて、JavaScriptやWebに関する技術を基礎からやさしく解説した入門書です。全部で14章に分かれており、各章でJavaScriptの特定のテーマについて解説します。読み終えるころには、JavaScriptのプログラムを作るために最低限必要な知識が身についていることでしょう。

各章には以下のような要素があり、理解を助けます。

1. 章の内容をイラストで紹介

各章の冒頭には内容をイラストで紹介するコーナーがあります。どんなことを学ぶのか、事前に把握して心の準備をしてください。

2. 本編の解説

初めての方でも理解できるよう、難しい言葉はなるべく使わずに丁寧に説明しています。

3. たくさんの図解

文章による説明の理解を助けるために図解を使って補足しているので、イメージをつかみやすくなっています。

4. 用語解説

用語解説

用語

必要に応じて、専門用語が登場します。その都度、解説していますが、補足情報などがある用語については、この要素で説明します。

5. Note

Note

説明に関連して、留意していただきたいことなどをまとめています。

6. コラム

コラムタイトル

Column

説明の流れからは少し外れますが、今後のために知っておいたほうがよい情報などをまとめています。

7. 注意

注意

やってはいけないことや、間違いのもとなど、注意が必要な内容についてまとめています。

8. Point

Point!

▶ **解説してきたことを、再度確認するために、簡潔にまとめています。見直しなどをする際に活用してください。**



9. 例

各章では、実際に動くサンプルプログラムを例にして解説しています。“リストX.1”などとキャプションが付いているものは、翔泳社のWebページからダウンロードすることができます。ダウンロードの詳細は、後掲の「サンプルのダウンロードについて」をお読みください。

なお、紙面の都合上、コードが折り返してあるところは、↵で示しています。実際のコードでは↵を削除して、次の行とつなげて書いてください。

10. チェックテスト

節末や章末にはチェックテストを用意しています。理解度をはかるために、ぜひチャレンジしてみてください。

本書の対象読者・前提知識

本書は、プログラミングをやったことがない初心者を対象としています。必要な知識はPCの基本操作だけです。学生の方から社会人、一度はプログラミングを始めたものの挫折してしまった方まで、幅広くお使いいただけます。

学習の進め方

本書は、「プログラミング経験がない方でも無理なく読み進められる」「電車の中などで読みながら理解できる」ということを1つの目標として執筆しました。わかりにくい概念やプログラムの実行イメージなどをイラストや図解を活用して説明しているため、一通りお読みいただくだけでもJavaScriptを体験できるでしょう。ただし、プログラムを自作できるようになる近道は、やはり「プログラムを書くこと」です。そのため、本書を「読む」だけでなく「プログラムを書く」こともおすすめします。

JavaScriptは、プログラミング言語としては珍しく、開発環境を自分で構築する必要がありません。横着な人でも今日からすぐにプログラミングを始めることができます。本書では、「プログラムを書く」ことも想定し、すべての項目ごとにサンプルプログラムを掲載しています。実際にプログラムを自分の手で入力し、動かしてみてください。これがなにより大切です。プログラミングを実体験することで、学習内容の理解がより一層深まることでしょう。

第1章～第3章▶ JavaScriptを学ぶ上での事前知識です。JavaScriptの概要と、JavaScriptを学ぶにあたって必要なHTML/CSSの基礎的な知識を紹介しています。Web制作に関する知識のある方は、読み飛ばしていただいてかまいません。

第4章～第7章▶ JavaScriptの基礎文法を学びます。動くWebページを作るわけではないので少々退屈に感じるかもしれませんが、ここでは早く先に進みたい気持ちをグッとこらえて、まずはしっかりと基盤を固めましょう。他のプログラミング言語の知識のある人は、軽く目を通す程度で良いでしょう。

第8章～第10章▶ JavaScriptを使って動きのあるWebページを作ります。入力フォームの状態を操作したり、ボタンがクリックされたタイミングでHTMLを書き換えたりします。実用的なサンプルプログラムも多数掲載していますので、この章を学び終わった頃には実際にWebサイトを制作するイメージがわいてくることでしょう。

第11章～第14章▶ JavaScriptのライブラリであるjQueryを利用して、アニメーションやAjaxなどのより高度な技術について学びます。JavaScriptを実

務で利用する上では欠かせない知識です。特に、今後Webプログラミングを仕事にしたいと思っている方にとっては必修の内容となります。

サンプルのダウンロードについて

本書に掲載したサンプルプログラムのソースコードは、翔泳社のWebページからダウンロードできます。以下のURLにアクセスして、リンクをクリックしてください。

サンプルのダウンロード

▶ http://www.shoeisha.co.jp/book/download/9784798125831

また、シリーズの専用ページを用意しています。このWebページでは、本書の補足資料（サンプル集）などを掲載する予定です。

翔泳社の「スラスラわかる」シリーズ Webページ

▶ http://www.shoeisha.co.jp/book/surasura/

サンプル集について

上記のWebページでは、Webサイトの制作でよく使われている小技的なJavaScriptのサンプル集をPDFで配布しています。本書の学習の総仕上げとして活用してください。

サンプルプログラムの動作環境について

サンプルプログラムを実行するには、ブラウザが必要です。本書では、Google Chromeを使って解説していますが、他のブラウザでも実行できます。JavaScriptで必要な環境については1.3節、サンプルプログラムの実行方法については2.3節および4.1節を参照してください。

なお、動作確認は以下の環境で行ないました。

OS ● Microsoft Windows 7/8（64ビット）

開発環境 ● Google Chrome、Microsoft Internet Explorer 8/9

はじめに …… iii
本書について …… iv
学習の進め方 …… vi
サンプルのダウンロードについて …… vii
サンプルプログラムの動作環境について …… vii

第1章 基礎知識と準備 …… 1

1.1 JavaScriptとWebページの基礎知識 …… 3

JavaScriptとは …… 3
Webページを作成するための技術 …… 6
静的なWebページと動的なWebページ …… 7

1.2 Webアプリケーションの仕組み …… 10

インターネットの仕組み …… 10
Webアプリケーションとスクリプト …… 11
クライアントサイドスクリプトのデメリット …… 12

1.3 JavaScriptの開発環境 …… 14

テキストエディタ …… 14
ブラウザ …… 17

第2章 HTML …… 21

2.1 HTMLとは …… 23

Webページの実体 …… 23

2.2 HTMLの書き方 …… 26

HTMLの注意点 …… 26
基本的な書き方 …… 26
HTMLの構成 …… 28

2.3 HTMLの作成と表示 …… 30
簡単なHTMLを作ろう！ …… 30
タグの意味 …… 33

2.4 主要なタグ …… 35
<body>要素の中に記述する要素 …… 35
リンクの利用 …… 36
画像の表示 …… 40

2.5 特殊なタグ …… 42
複数組み合わせて1つの部品を生成するタグ …… 42
リスト …… 42
テーブル …… 44
フォーム …… 45
セレクトボックス …… 49

第3章 CSS 53

3.1 CSSの基礎知識 …… 55
CSSとは …… 55
CSSを記述する場所 …… 56

3.2 CSSの書き方 …… 58
セレクタとプロパティの指定 …… 58

3.3 セレクタとプロパティ …… 60
プロパティの種類 …… 60
セレクタの種類 …… 61

第4章 基本文法 69

4.1 書き方の基本 71
まずは1行のJavaScriptから！ 71
書き方のルール 72
JavaScriptを記述する場所 73
コメント 76
値の出力 77

4.2 変数とデータ型 83
変数とは 83
変数の宣言 84
変数への値の代入 86
変数の表示 87
データ型 88

4.3 演算子 90
演算子とは 90
算術演算子 90
複合代入演算子 91

4.4 配列 94
配列とは 94
配列は要素の集合体 94
配列の書き方 95
二次元配列 99

4.5 連想配列 102
連想配列とは 102

第5章 制御構文 107

5.1 if文 …… 109

分岐処理とは …… 109

条件式の書き方 …… 110

より複雑な条件式の書き方 …… 112

多方向分岐 …… 115

5.2 switch …… 118

switch文 …… 118

5.3 while文 …… 122

反復処理とは …… 122

5.4 for文 …… 125

for文とは …… 125

for文の応用例 …… 127

for-in文 …… 128

5.5 繰り返しの制御 …… 132

繰り返しの中のbreak文 …… 132

continue文 …… 133

第6章 関数 139

6.1 関数〜自作関数 …… 141

関数とは …… 141

引数も戻り値もない関数の書き方 …… 142

引数のある関数の書き方 …… 144

戻り値のある関数の書き方 …… 145

6.2 組み込み関数 ……149
組み込み関数 ……149
数値変換に関する関数 ……149
エンコード／デコードに関する関数 ……151

6.3 変数のスコープ ……153
スコープとは ……153

6.4 関数の応用 ……159
関数の代入と匿名関数 ……159
即時関数 ……161

第7章 オブジェクトの基礎 165

7.1 オブジェクト、メソッド、プロパティ ……167
オブジェクトとは ……167
thisキーワード ……169
別のオブジェクトのメソッドを利用する ……169

7.2 組み込みオブジェクト ……173
組み込みオブジェクトとは ……173
Objectオブジェクト ……173
Dateオブジェクト ……174
Mathオブジェクト ……175
Arrayオブジェクト ……176
Stringオブジェクト ……178

7.3 正規表現 ……182
正規表現とは ……182
正規表現の書き方 ……183
正規表現の利用 ……186

第8章 イベント 189

8.1 イベント駆動型モデル ……191
イベントとは ……191
主なイベント ……191

8.2 イベントの使い方 ……193
clickイベント ……193
loadイベント ……195
mouseover／mouseoutイベント ……197
focus／blurイベント ……198
onchangeイベント ……198

8.3 イベント発生元の特定 ……200
thisキーワードの利用 ……200

第9章 ブラウザオブジェクト 205

9.1 ブラウザオブジェクト ……207
ブラウザオブジェクトとは ……207

9.2 Windowオブジェクト ……210
ブラウザのウィンドウ全体の情報 ……210
confirmメソッド ……210
openメソッドとcloseメソッド ……212
setIntervalメソッドとclearIntervalメソッド ……215

9.3 LocationオブジェクトとHistoryオブジェクト ……218
WebページのURL情報（Location） ……218
Webページの履歴情報（History） ……218

9.4 Documentオブジェクト 221
Webページの表示情報 221

第10章 DOM 227

10.1 DOM 229
DOMとは 229

10.2 DocumentオブジェクトによるHTML要素の操作 231
要素へのアクセス（要素の取得） 231
要素の内容の操作 237

10.3 イベントハンドラの設定 241
JavaScriptによる記述方法 241
引数のある関数をイベントハンドラに設定する 246

第11章 jQueryの基礎知識 251

11.1 jQueryとは 253
クロスブラウザ問題 253
jQueryのメリット 254

11.2 jQueryの利用準備 257
入手方法 257
事前にダウンロードしてHTMLファイルに読み込む 257
GoogleのサーバーからHTMLファイルに直接読み込む 259

11.3 jQueryプラグイン 261
プラグインとは 261

プラグインの使い方（LightBox系プラグイン）……261

第12章 jQueryの文法 267

12.1 jQueryの基本の書き方……269
readyメソッド……269

12.2 セレクタ……271
要素の取得……271
階層セレクタ……273

12.3 要素の抽出……279
フィルタ……279
要素の絞り込み……284

12.4 内容と属性の操作……291
内容と属性の変更……291
イベントハンドラの設定……295
CSSクラスの変更……302

12.5 要素の挿入……307
要素の追加／削除／置換……307
DOM操作の応用……312

第13章 Ajax 317

13.1 Ajaxの基礎……319
Ajaxとは……319
AjaxによるHTMLの読み込み……321

13.2 JSONとJSONP …… 325
JSONとは …… 325
JSONの書き方と取得方法 …… 326
JSONPとは …… 329
Web APIとは …… 330
JSONPの利用 …… 330

第14章 アニメーション 335

14.1 表示状態の切り替え …… 337
アニメーションとエフェクト …… 337
左上を基点に切り替える〜show／hide／toggleメソッド …… 337
上下にスライドする〜slideDown／slideUpメソッド …… 340
ぼんやり透けていく〜fadeIn／fadeOutメソッド …… 342

14.2 独自アニメーションの作成 …… 345
animateメソッドによるアニメーション …… 345
animateメソッドに対するメソッドチェーン …… 347
スライドショーの作成 …… 348
スライドショーの解説〜HTMLとCSS …… 350

次のステップへ …… 358

Appendix 付録 プログラムがうまく動かないときの対処法〜デバッグ 361

A.1 プログラムのバグを取り除こう …… 362
バグの要因 …… 362

A.2 スクリプトの記述ミス（構文エラー）の場合 …… 363
A.3 意図したとおりに動かない場合 …… 365
ブレークポイントの設定 …… 367
スクリプトのステップ実行 …… 368
チェックテストの解答例 …… 374
索引 …… 392

Column

●Webサイトと Webページ …… 5
●メモ帳の起動方法 …… 15
●ファイルの拡張子を表示する設定（Windows の場合） …… 18
●Internet Explorer 9でWebページとコンソールを表示する場合 …… 80
●オープンソースライセンスについて …… 265
●ステップ実行以外の各種アイコンの機能 …… 373

第1章

基礎知識と準備

この章で学ぶこと

- JavaScriptとは
- Webアプリケーションの仕組み
- JavaScriptの開発環境



● JavaScriptでなにができるの？ ●



まずは学習にあたって、JavaScriptとはどのようなものか、また、JavaScriptを取り巻く周辺技術など、事前に知っておくべき基礎的な事柄について見ていきましょう。



1 JavaScriptとWebページの基礎知識

JavaScriptとは

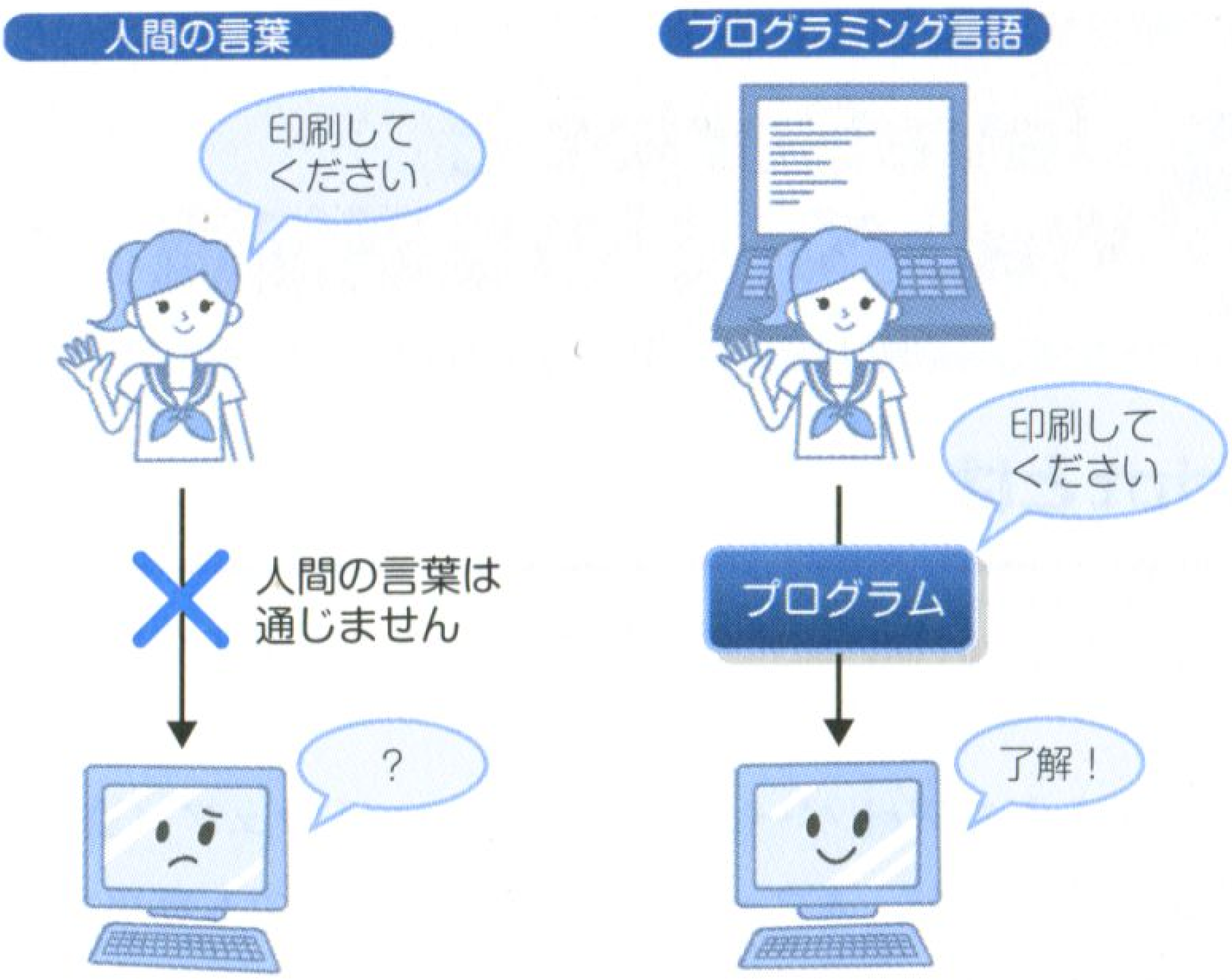
本書のテーマであるJavaScriptは、主に**ブラウザ**上で動作する**スクリプト言語**です。

ブラウザはインターネット上のWebページを表示するソフトウェア（道具）です。ふだんあまり意識しませんが、スマートフォンやケータイでも専用ブラウザでWebページを表示しています。



さまざまな種類のブラウザがある

では、スクリプト言語とはなんでしょうか？　スクリプト言語は、**プログラミング言語**の一種です。プログラミング言語とは、パソコンに対してなんらかの作業を命令するための言葉です。プログラミング言語で記述した（書いた）命令文を**プログラム**と呼びます。パソコンは、人間の言葉を理解することができません。しかし、プログラミング言語でプログラムを記述すれば、パソコンに思いどおりのことをやらせることができるのです。



プログラミング言語はパソコンにやらせたい作業を伝えるための言葉

ちなみに、プログラムを記述することを**プログラミング**と呼びます（プログラムを作成する／開発するとも言います）。

さまざまな種類のプログラミング言語がありますが、スクリプト言語はその1つで、簡易プログラミング言語とも呼ばれています。「簡易」とつくことからもわかるように、一般的なプログラミング言語よりも簡易的な文法で記述できるのがスクリプト言語の特長です。プログラムを記述する際、記述内容がわかりやすく、命令文の記述量も少なくて済むため、プログラミング初心者でも習得が容易というメリットがあります。

また、スクリプト言語で記述したプログラムは**スクリプト**とも呼びます。

Note 以降、本書では、JavaScriptで記述したプログラムはスクリプトと表記します。

JavaScriptは、ブラウザに対してなんらかの作業を命令するためのスクリプト言語です。JavaScriptで記述したスクリプトを受け取ったブラウザは、その命令文を実行します。



JavaScriptはブラウザに作業を命令する

ブラウザに命令できる作業は、Webページ上に表示するさまざまなコンテンツ（情報）の操作です。つまり、JavaScriptでスクリプトを記述することで、「動き」のあるWebページを作成できます。たとえば、次のようなことがJavaScriptで実現できます。

- Webページにアニメーション効果を追加する（色や形の変更、スクロールなど）
- Webページを切り替えずに表示の一部だけを更新する（置き換える）
- リアルタイムにWebページ上の入力値チェックや計算を行なう
- Webページ上のボタンクリックでウィンドウを開閉する

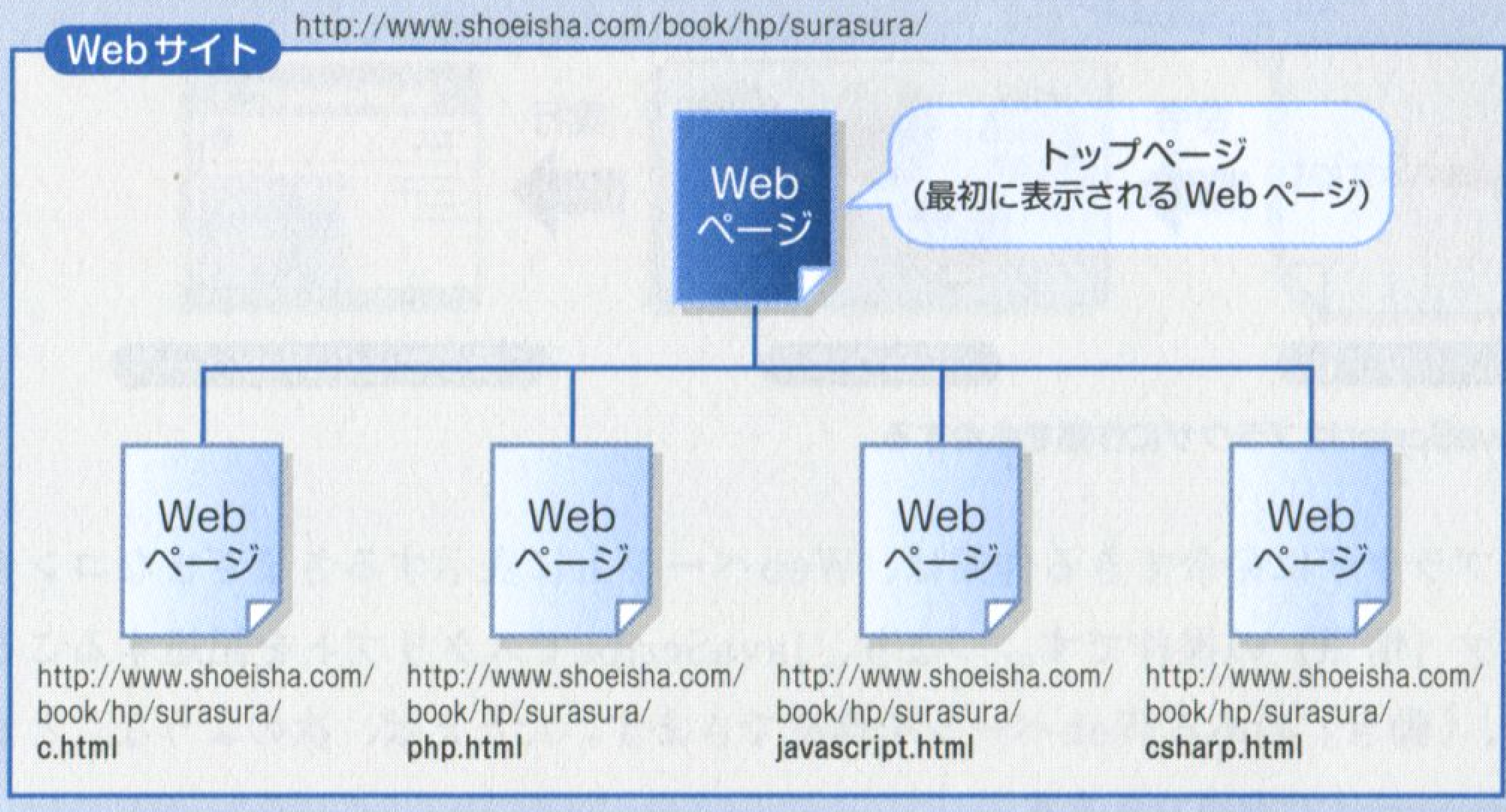
Note

JavaScriptは、よくプログラミング言語のJavaと混同されることがありますが、JavaとJavaScriptはまったくの別物です。JavaScriptは、当初LiveScriptという名前でしたが、Javaの人気にあやかってJavaScriptに名称が変更されました。

Column

Webサイトと Webページ

「Webサイト」と「Webページ」という言葉は、同じ意味で使われることもありますが、厳密には使い分けます。よく、Webサイトは"一冊の本"、Webページは"本の1つ1つのページ"にたとえられます。通常、複数のWebページが集まってWebサイトが作られているからです。以下は、本書の公式Webサイトのページ構成です。Webサイトをブラウザで表示したとき、最初に表示されるWebページのことを「トップページ」と呼びます。



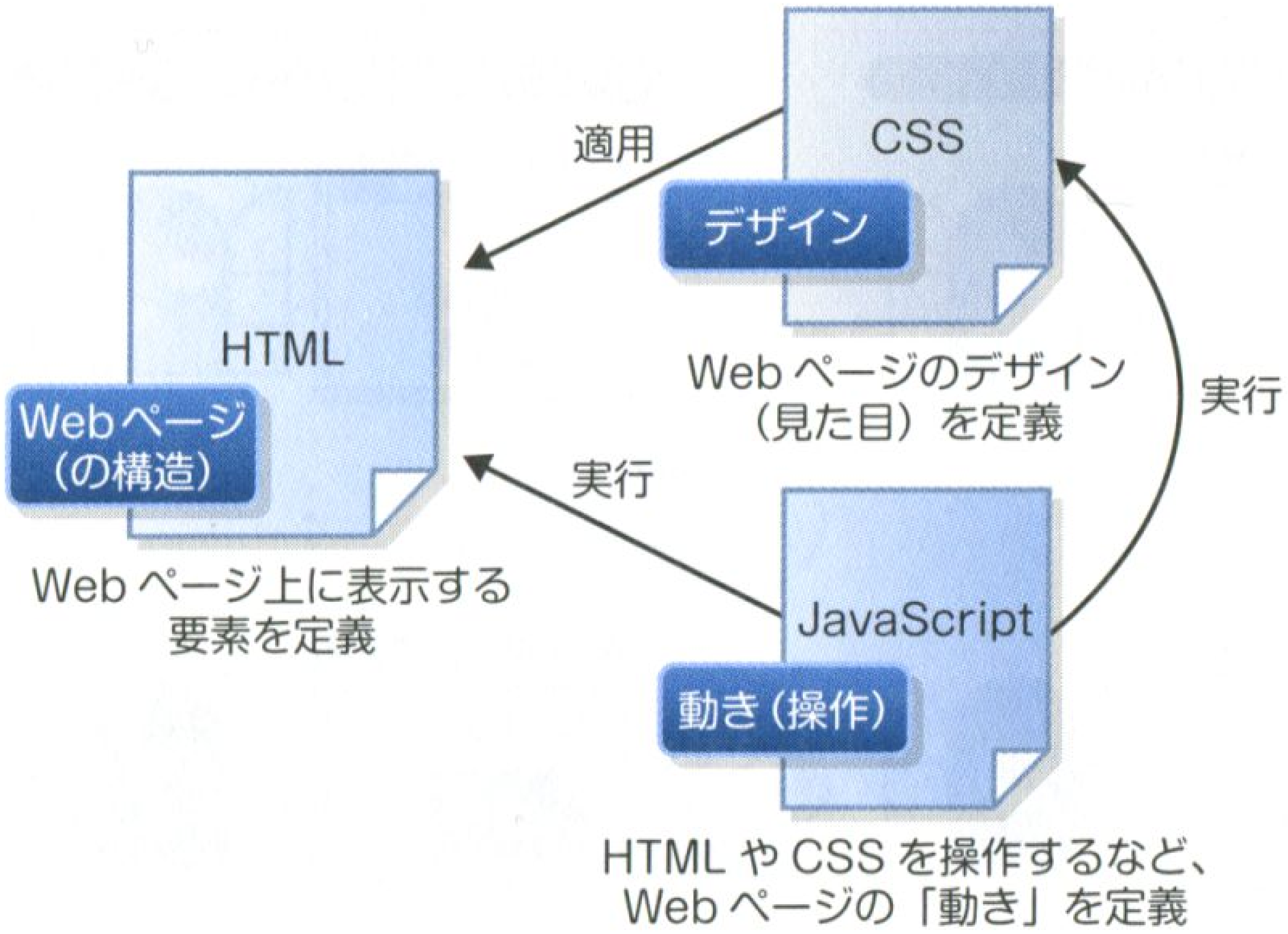
Webサイトは複数のWebページの集合体

ちなみに、Webサイトは「サイト」、Webページは「ページ」とも言います。なお、WebサイトやWebページを「ホームページ」と呼ぶ場合がありますが、本来、ホームページとは、ブラウザを起動したとき、あるいはブラウザのホームボタンを押したときに表示されるWebページのことを指します（これは「スタートページ」とも言います）。

Webページを作成するための技術

Webページの作成にJavaScriptを利用する場合、関連技術の知識も必要不可欠です。Webページの作成には、主に次の3つの技術が用いられます。

- HTML（エイチティエムエル）（HyperText Markup Language）：タイトルや見出し、文章、画像など、Webページ上に表示するコンテンツを定義する言語。
- CSS（シーエスエス）（Cascading Style Sheets）：コンテンツの色や配置など、Webページのデザイン（見た目）を定義する言語。
- JavaScript：Webページのコンテンツ（HTML）やデザイン（CSS）を操作するなど、Webページの「動き」を定義する言語。



Webページの作成に必要な技術

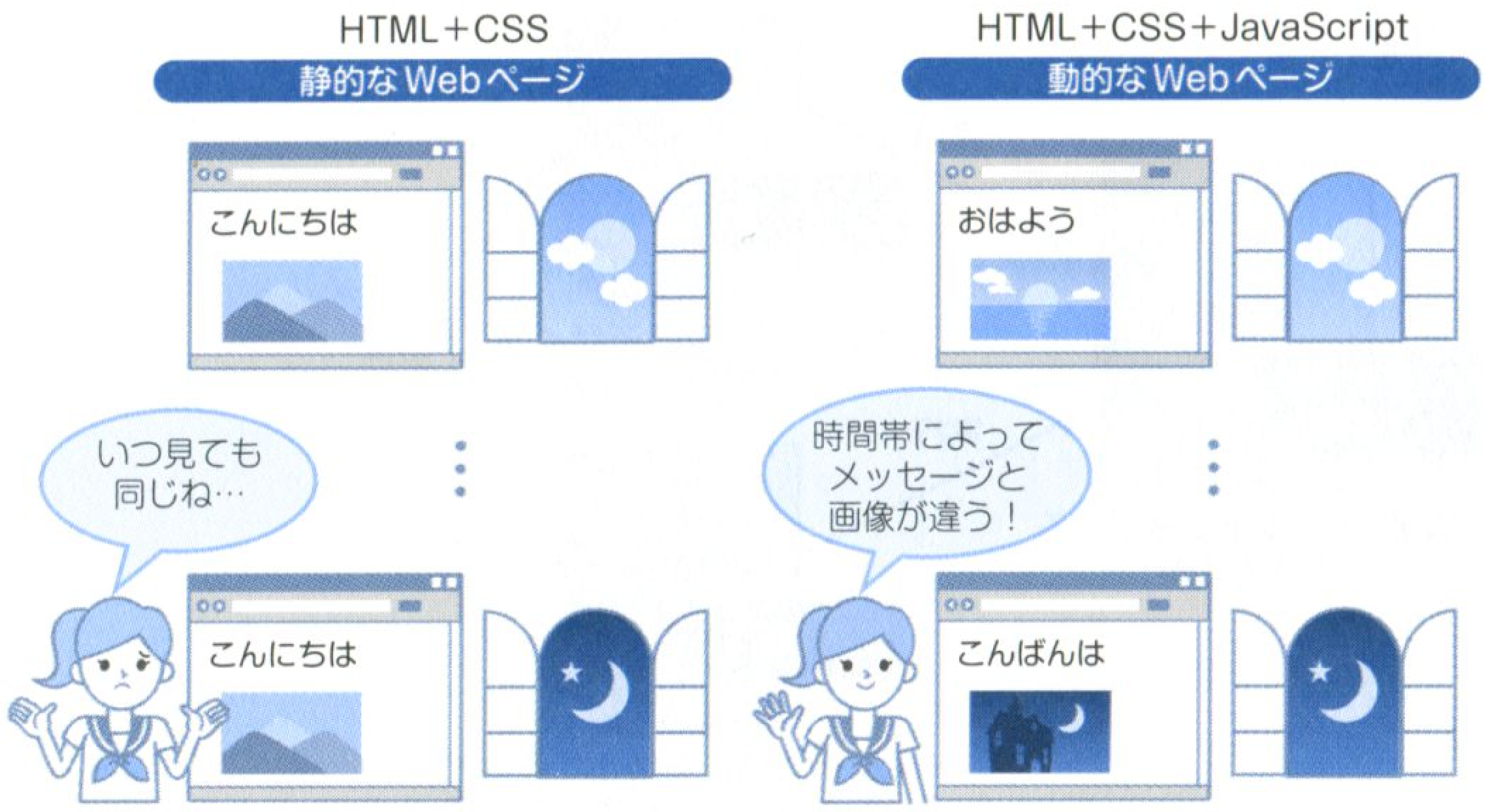
この3つがWebページを作成するための基本セットです。次の点に注目してください。

- HTMLでWebページを作成して、CSSでデザインを適用し、JavaScriptでWebページの要素やデザインを操作する

よって、JavaScriptを習得し、かつ使いこなすには、HTMLやCSSの基礎的な知識が必要になります（HTMLとCSSについては第2章と第3章で詳しく解説します）。

静的なWebページと動的なWebページ

続いて、JavaScriptと関わりの深いWebページの概念について学びましょう。Webページには、**静的**（せいてき）**なWebページ**と**動的**（どうてき）**なWebページ**の2種類があります。



静的なWebページと動的なWebページの違い

JavaScriptはもともと、Webページに「動き」を持たせるために作られた言語です。JavaScriptが普及する前は、HTMLとCSSによる、あらかじめ決められた文章や画像をユーザーに見せるだけのWebページがほとんどでした。つまり、同じURLにアクセスした場合、毎回同じ内容やデザインが表示されるWebページです。これを静的なWebページと呼びます。

これに対して、動的なWebページは、同じURLであっても異なる内容やデザインが表示されます。たとえば、よくあるのが次のようなWebページです。

- 10秒おきにトップニュース一覧が入れ替わる
- ページ全体のテーマカラーをユーザーが自由に選択できる
- マウスカーソルが画像の上に乗ると別の画像に切り替わる
- 1つ目のセレクトボックスから都道府県を選択すると、2つ目のセレクトボックスで市区町村を選択できるようになる

どれもWebページでよく見る表現ですよね。JavaScriptは現在、Webページ上で多彩な表現を実現するために欠かせない技術となっています。

Point!

- パソコンの世界での「言語」とは、人がパソコンに命令するための言葉。プログラミング言語でパソコンへの命令文（プログラム）を記述する。
- JavaScriptとは、ブラウザに作業を命令するためのプログラミング（スクリプト）言語。
- JavaScriptを利用すると、「動き」のあるWebページを作成できる。
- Webページには、静的ページ、動的ページの２種類がある。



Check Test 1

Q1 以下はWebページの制作に利用する主な技術の説明です。空欄A～Cを埋めてください。

技術	説明
A	タイトルや見出し、文章、画像など、Web ページ上に表示するコンテンツを定義する言語
B	ページ要素の色や配置など、Web ページのデザインを定義する言語
C	Web ページの要素やデザインを操作するなど、Web ページの「動き」を定義する言語

Q2 以下はプログラミング言語とスクリプト言語の説明です。空欄A～Dを埋めてください。

用語	説明
プログラミング言語	[A] に対してなんらかの作業を [B] するための言語
スクリプト言語	プログラミング言語の一種だが、プログラミング言語よりも [C] な文法で記述できるという特長を持つ。別名 [D] とも呼ばれる

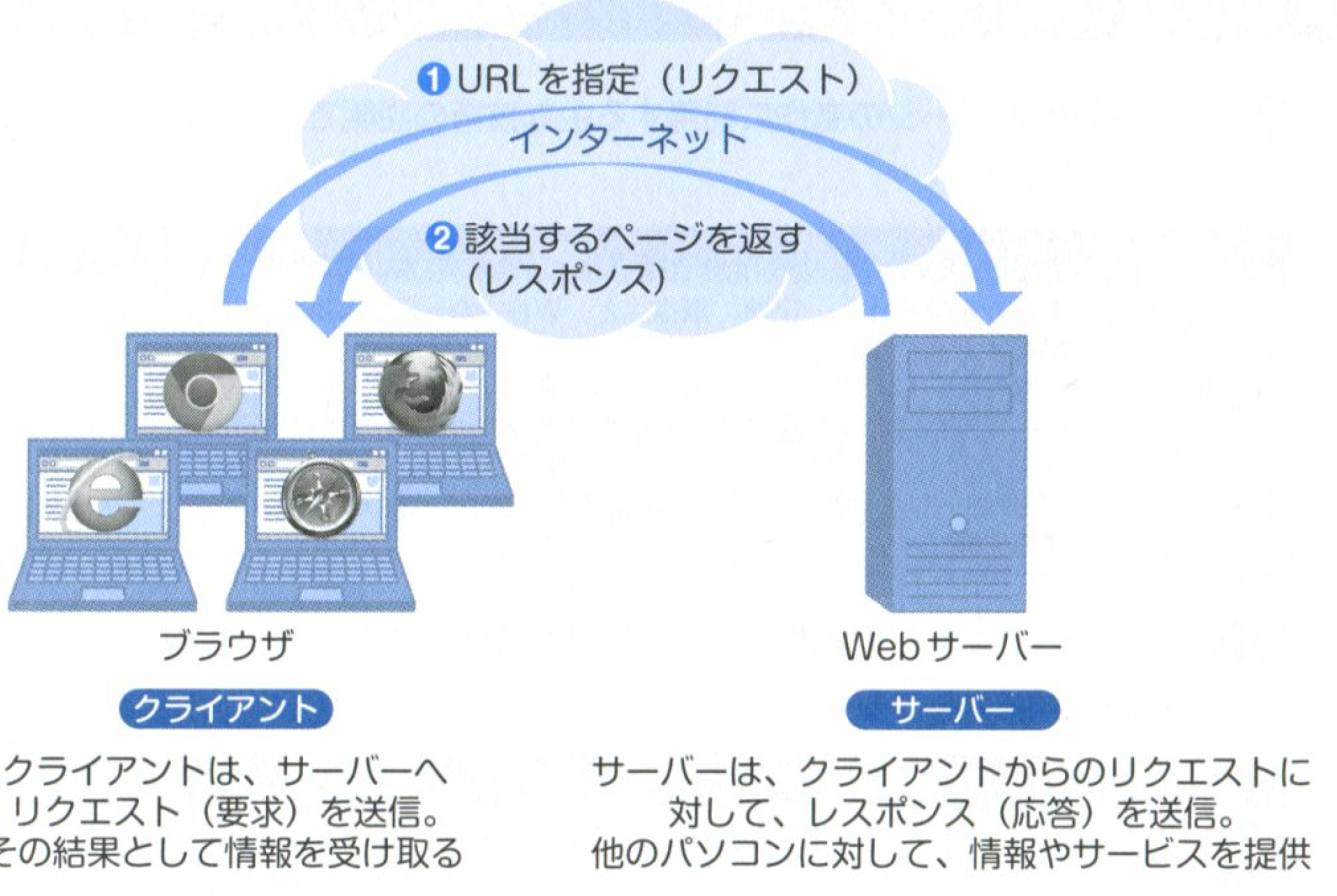
Q3 静的なWebページ、動的なWebページとはなんですか？

解答は巻末に掲載

1-2 Webアプリケーションの仕組み

インターネットの仕組み

続いて、これまたWebページを作るときには欠かせない、**サーバー**と**クライアント**という概念について学びましょう。次の図は、インターネットの仕組みを簡単に表わしています。



インターネットの仕組み

Webサーバーとは、Webサイトをインターネットに公開するためのソフトウェアを搭載したパソコンのことです。クライアントは、各ユーザーのパソコンに搭載されたブラウザです。

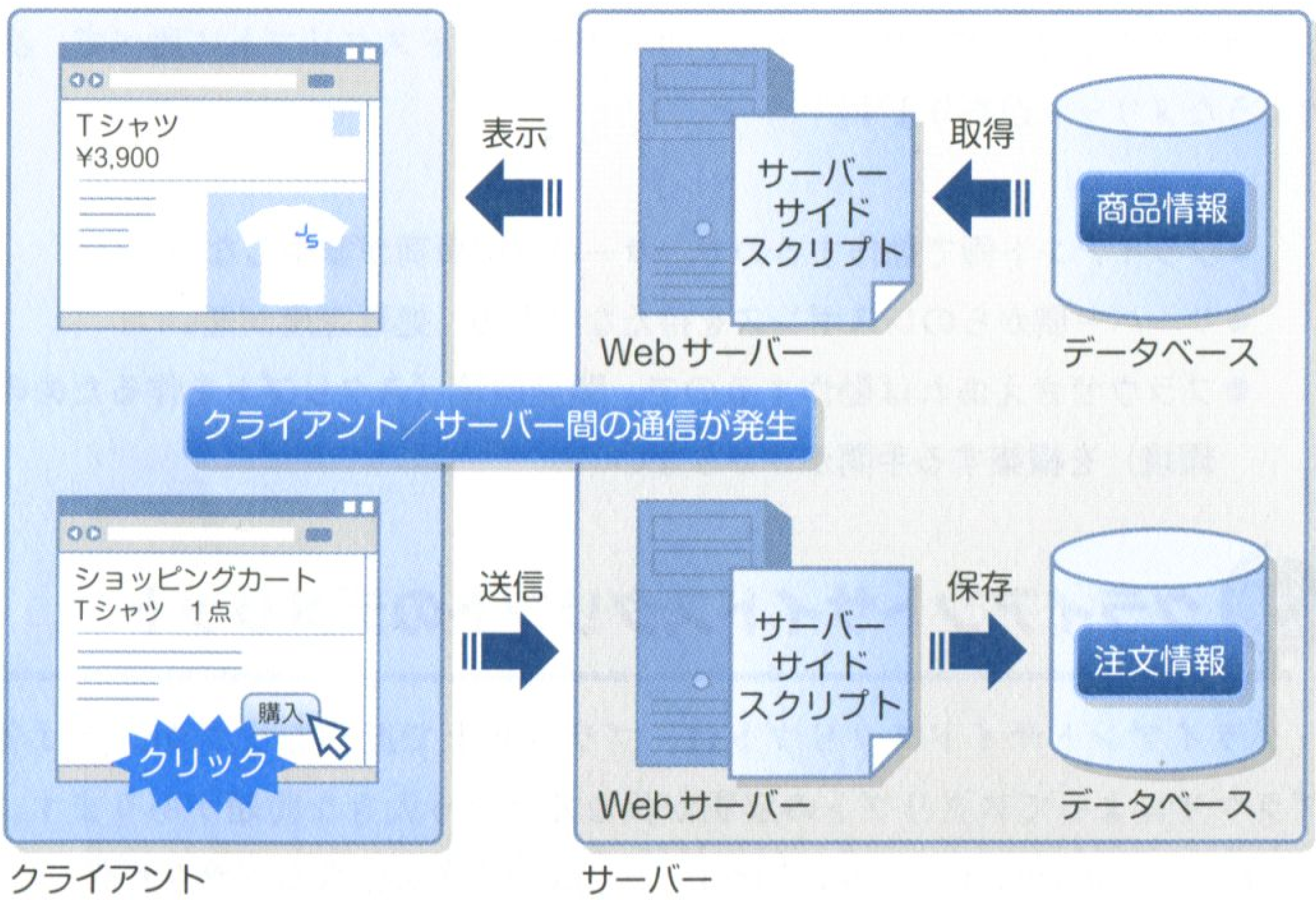
WebサーバーがWebサイトを公開すると、サイト内の各ページにはそれぞれインターネット上の住所であるURLが割り振られます。このURLにクライアントからアクセスすることで、該当するWebページを見ることができる仕組みになっています。

Webアプリケーションとスクリプト

Webサイトの中でも、ショッピングサイトなどの複雑な機能を持ったWebサイトのことを**Webアプリケーション**と呼びます。たとえば、GoogleやYahoo!などの検索サイト、Amazonや楽天市場などのショッピングサイト、GmailやYahoo!メール、どれもWebアプリケーションです。Webアプリケーションは、一般的に複数のプログラミング言語やスクリプト言語を組み合わせて作られています。

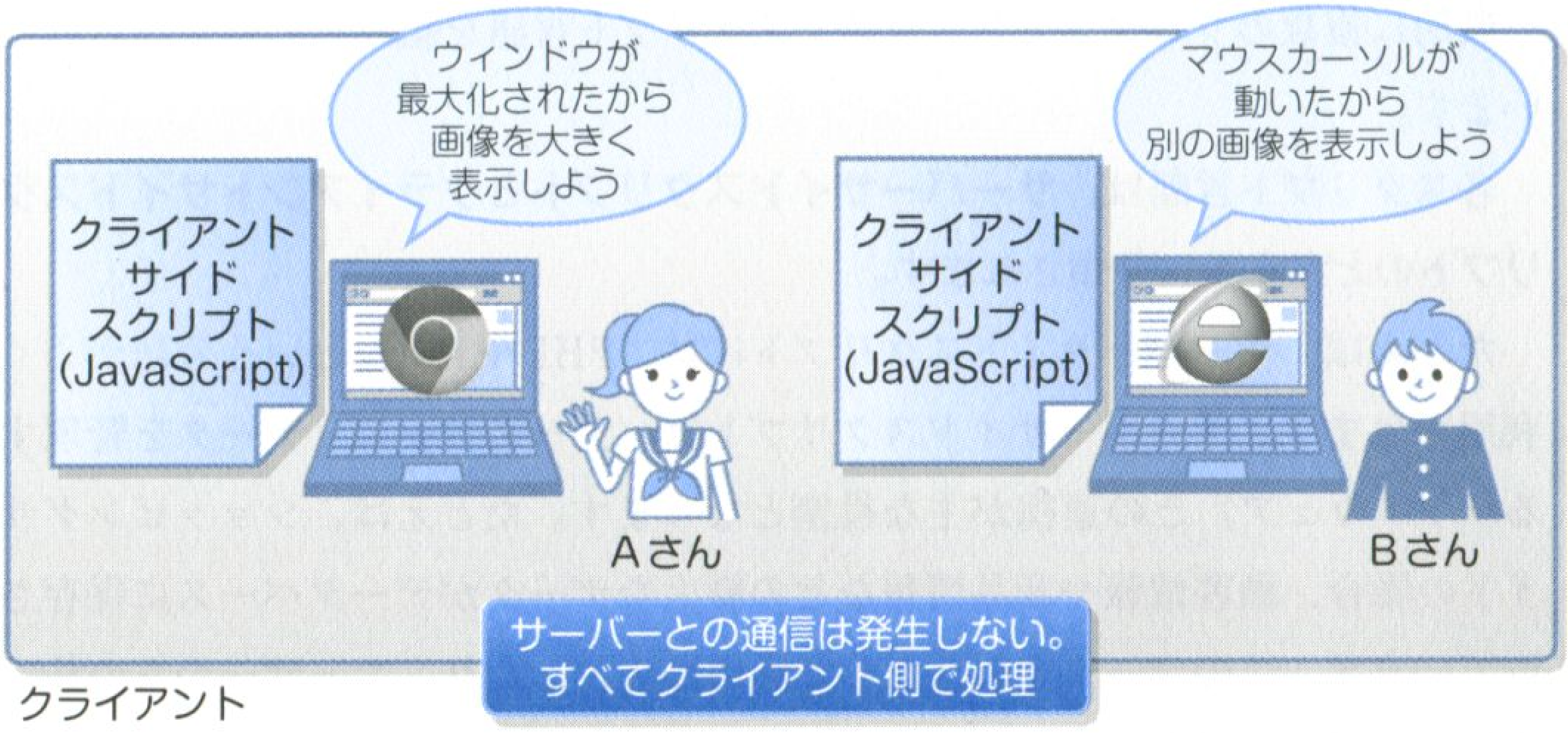
各スクリプト言語は、**サーバーサイドスクリプト**と**クライアントサイドスクリプト**のどちらかに分類されます。

たとえば、サーバーサイドスクリプトには、PHPやRubyという言語がよく利用されます。サーバーサイドスクリプトは、データベース（データを管理するソフトウェア）との連携が主な役割となります。たとえば、ショッピングサイトの場合、顧客情報や商品情報などの膨大なデータがデータベースに保存されています。このデータを引き出して画面に表示したり、ユーザーからの注文データを新たに保存したりするのがサーバーサイドスクリプトの役割です。



サーバーサイドスクリプトの役割

それに対してクライアントサイドスクリプトは、そのシェアのほとんどをJavaScriptが独占しています。クライアントサイドスクリプトは、Webページの操作が主な役割になります。マウスカーソルやブラウザのウィンドウサイズなど、各クライアントの状態を見て、それに合わせた表示方法に切り替えることができます。



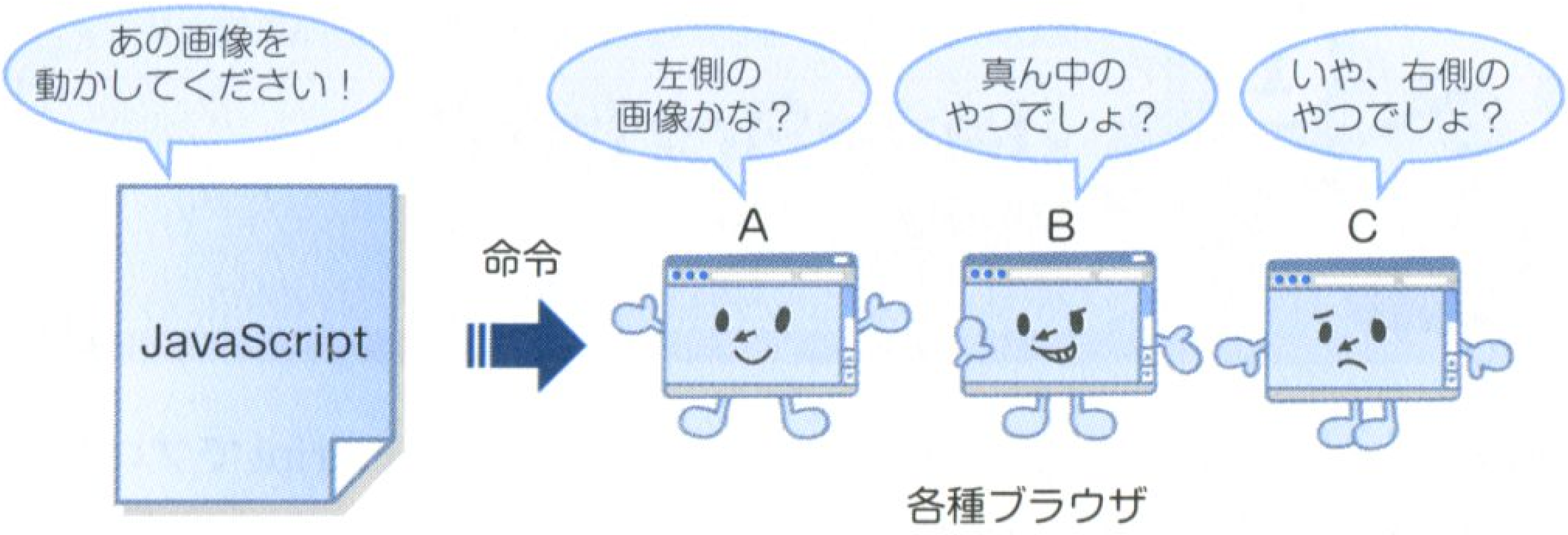
クライアントスクリプトの役割

クライアントサイドスクリプトは、サーバーサイドスクリプトに比べて、次のようなメリットがあります。

- クライアント側で動作するので、サーバーに負荷が集中しない
- サーバー側からのレスポンスを待たないため、処理速度が速い
- ブラウザさえあれば動作するので、開発環境（スクリプトを作るための環境）を構築する手間がかからない

クライアントサイドスクリプトのデメリット

クライアントサイドスクリプトは、ブラウザ上で動作しますが、実は各ブラウザによってスクリプトの解釈が異なるという大きな問題があります。Internet Explorer（インターネット エクスプローラー）、Chrome（クローム）、Firefox（ファイアフォックス）、Opera（オペラ）などの主要なブラウザ間でも、スクリプトの実行結果に微妙な違いが発生します。



ブラウザによって命令の解釈が異なる

この問題の解決方法として、JavaScriptのライブラリ（ある機能をまとめたもの）である**jQuery**（ジェイクエリー）の利用が挙げられます。jQueryを使うことで、ブラウザ間の差異を吸収し、まったく同じ動作をさせることが可能となります。

> **Note** jQueryは、JavaScriptを手軽に利用できるようにするライブラリです。jQueryを使うと、Webページに対する複雑な操作もより簡単に記述できるようになります。jQueryの詳しい使い方は第11章以降で学びます。

Point!

- Webアプリケーションは、サーバーサイドスクリプトとクライアントサイドスクリプトで構成される。
- ブラウザの種類やバージョンによってスクリプトの解釈が異なる場合がある。それを解消するためにJavaScriptのライブラリを使う。



■ Check Test 2

Q1 以下はクライアントとサーバーの説明です。空欄A～Dを埋めてください。

クライアント	サーバーへ［　A　］を送信。その結果として情報を受け取る。各ユーザーのパソコンに搭載された［　B　］のこと
サーバー	クライアントからの［　A　］に対して、［　C　］を送信。Webサイトをインターネットに公開するためのソフトウェアを搭載したパソコンを［　D　］と呼ぶ

Q2 サーバーサイドスクリプト、クライアントサイドスクリプトとはなんですか？

➲解答は巻末に掲載

1-3 JavaScriptの開発環境

最後に、JavaScriptの開発環境を準備しましょう。JavaScriptでプログラミングするときに最低限必要なのは、次の2つです。

- テキストエディタ
- ブラウザ

テキストエディタでスクリプト（プログラム）を書き、ブラウザで動作を確認（テスト）します。他のプログラミング言語と比べて、非常に簡単に開発環境が準備できるのもJavaScriptの特長です。

> **Note** テキストエディタは、パソコンでテキスト（パソコンの画面に表示可能な文字情報）のみのファイルを作成／編集／保存するためのツールです。ちなみに、テキストのみを記録したファイルのことをテキストファイルと呼びます。

テキストエディタ

Windowsならメモ帳、MacならTextEditが標準で付いていますが、JavaScriptのキーワードが色付きで表示される、高機能なものを利用すると良いでしょう。ふだん使っているものがあればそれでもかまいません。おすすめは、次のテキストエディタです（どちらもフリーソフトです）。

- TeraPad …………http://www5f.biglobe.ne.jp/~t-susumu/library/tpad.html
- サクラエディタ …http://sakura-editor.sourceforge.net/

ただし、あくまで学習用ということで、本書ではWindowsに装備されていて入手の手間がないメモ帳を使います。

Column

メモ帳の起動方法

Windows 8とそれ以外のOSで起動方法が異なります。ここでは、Windows 8とWindows 7の場合の起動方法を紹介します。メモ帳の利用時には、[書式(O)] メニューの [右端で折り返す(W)] を選択しておくと、プログラムが折り返されて見やすくなります。

●Windows 8の場合

まず次の手順に従って、デスクトップ画面の下部のタスクバーにメモ帳を追加します。

❶ Windows 8にログインします。

❷ スタートページのなにも表示されていない部分で、マウスを右クリックすると、画面下部に「すべてのアプリ」が表示されます。この「すべてのアプリ」をクリックします。



スタートメニュー下部の「すべてのアプリ」をクリック

❸ すべてのアプリが表示されるので、画面を右側へスクロールしていくと「メモ帳」の起動メニューがあります。



アプリページの「メモ帳」を右クリック

1

❹「メモ帳」を右クリックすると、画面下部に「タスクバーにピン留めする」が表示されるので、これをクリックします。



アプリページ下部の「タスクバーにピン留めする」をクリック

❺❸の画面に戻るので、今度は「メモ帳」をクリックします。すると、メモ帳が起動します。

これでデスクトップ下部のタスクバーにメモ帳のアイコンが追加されます。次回からは、このアイコンをクリックすればメモ帳を起動できます。

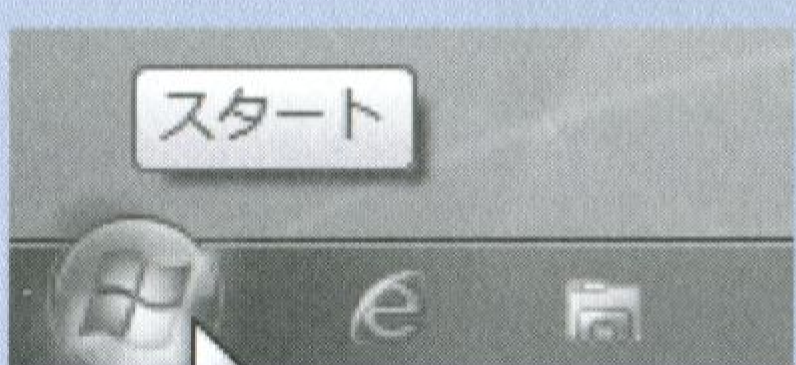


デスクトップ下部にメモ帳のアイコンが追加されている

●Windows 7の場合

まず次の手順に従って、Windowsのスタートメニューにメモ帳を追加します。

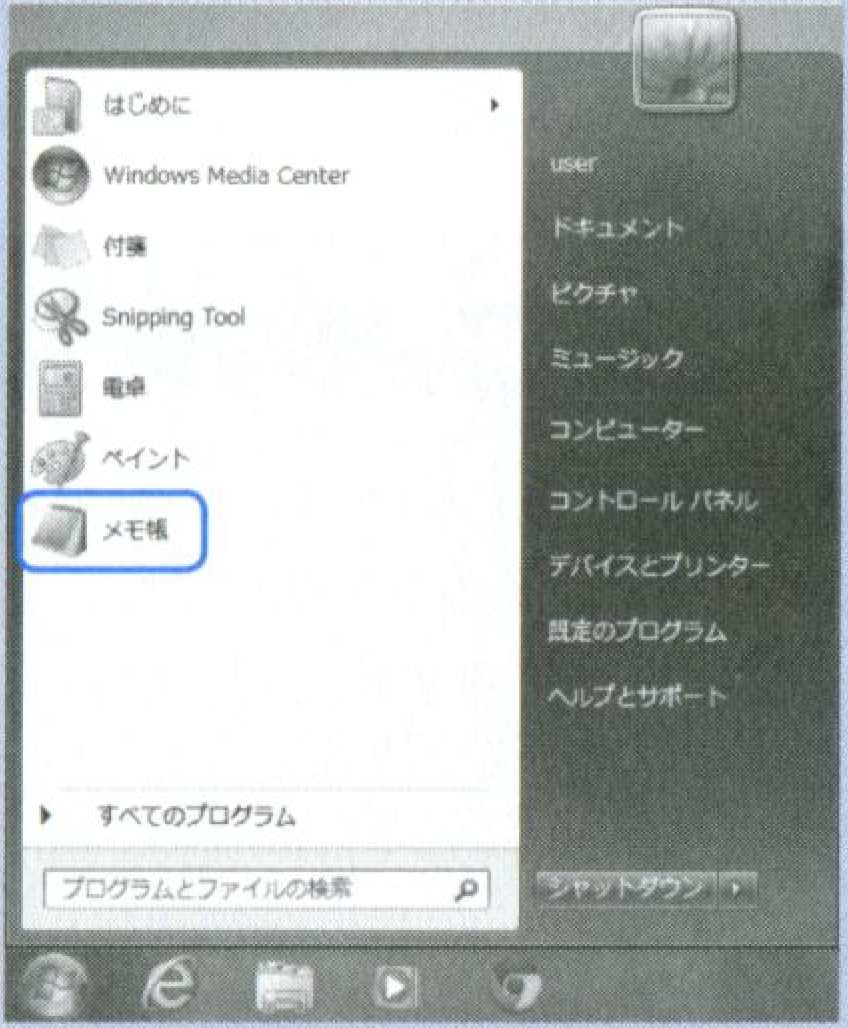
❶Windows 7にログインします。
❷デスクトップ画面の左下のスタートボタンをマウスでクリックし、さらに［すべてのプログラム］→［アクセサリ］の順でクリックします。



Windowsのスタートボタンをクリック

❸メニュー一覧の［メモ帳］を右クリックし、［スタートメニューに表示する］をクリックします。

これでWindowsのスタートメニューにメモ帳が追加されます。次回からは、デスクトップのスタートボタンをクリックして直接、メモ帳を起動できます。



スタートメニューにメモ帳が追加された

ブラウザ

ブラウザもふだん使っているものがあればそれでも良いですが、開発のしやすさから考えると、ChromeやFirefoxがおすすめです。FirefoxはFirebug（ファイアバグ）という開発用ツールを別途インストールする必要があるため、本書ではChromeを使います。下記のWebページからダウンロードしてインストールしておきましょう。

https://www.google.com/intl/ja/chrome/browser/

ただし、前節で触れたようにJavaScriptはブラウザによって動作結果が異なる場合があります。開発が終了し、実際にインターネットに公開する前には必ず他の主要ブラウザでもスクリプトが問題なく動作することを確認するようにしましょう。

表 1.1　主要ブラウザ一覧

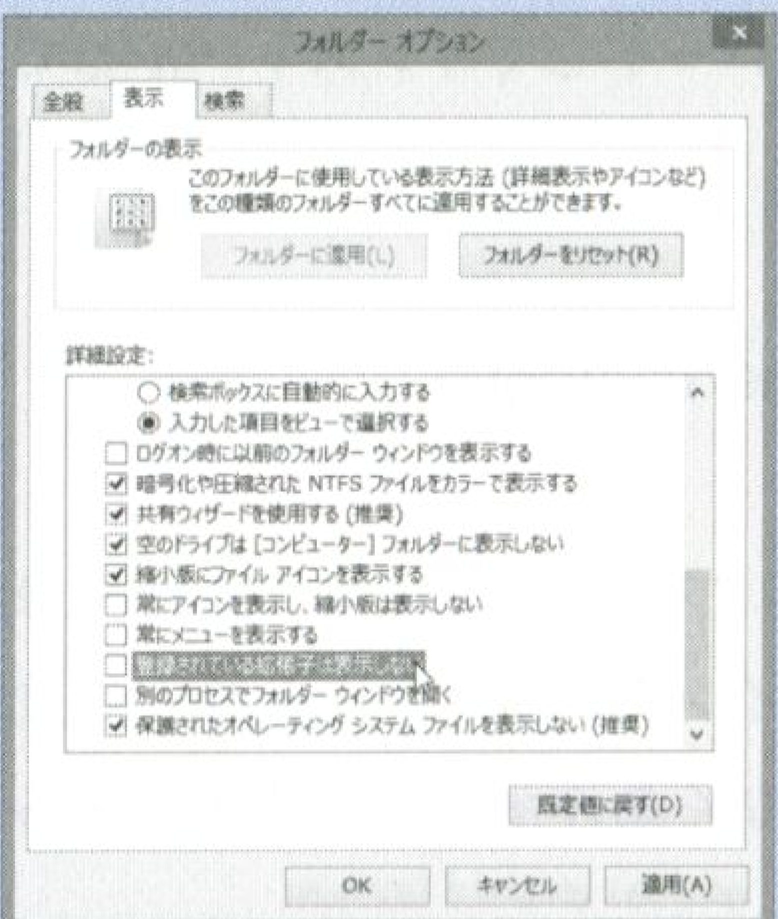
名前	入手先（公式 Web サイト）
Chrome	https://www.google.com/intl/ja/chrome/browser/
Internet Explorer	http://windows.microsoft.com/ja-JP/internet-explorer/products/ie/home
Firefox	http://www.mozilla.jp/firefox/
Opera	http://jp.opera.com/
Safari	http://www.apple.com/jp/safari/

Column

ファイルの拡張子を表示する設定（Windowsの場合）

本書では、ファイルの拡張子が表示されるように設定しています。拡張子とは、ファイル名の「.」とその右側の英数字のことで、ファイルの種類を表わす識別子です。本書を読み進める前に、次の手順に沿って拡張子を表示しておいてください。

❶ Windows 8の場合、スタート画面で「デスクトップ」をクリックします。Windows 7の場合、スタートボタンをクリックし「スタート」メニューを表示します。
❷「コントロールパネル」をクリックします。Windows 7の場合、さらに「デスクトップのカスタマイズ」をクリックします。
❸「フォルダーオプション」をクリックすると、「フォルダーオプション」ダイアログボックスが表示されます。
❹「表示」タブをクリックします。
❺「登録されている拡張子は表示しない」のチェックをはずします。
❻ [OK] ボタンをクリックします。



「フォルダーオプション」ダイアログボックス

Point!



▶ JavaScriptは、テキストエディタで記述し、ブラウザで動作を確認する。

この章の到達度チェック

Q1 JavaScriptとはなんですか？

Q2 JavaScriptの実行環境はなんですか？

Q3 以下はサーバサイドスクリプトと比較した、クライアントサイドスクリプトのメリットについての説明です。空欄A～Cを埋めてください。

- [A]で動作するので、[B]に負荷が集中しない
- [B]側からのレスポンスを待たないため、処理速度が[C]
- [D]さえあれば動作するので、開発環境を構築する手間がかからない

Q4 クライアントサイドスクリプトのデメリットにはどのようなものがありますか？

Q5 JavaScriptでプログラミングするときに最低限必要なものはなんですか？

➲解答は巻末に掲載

第2章

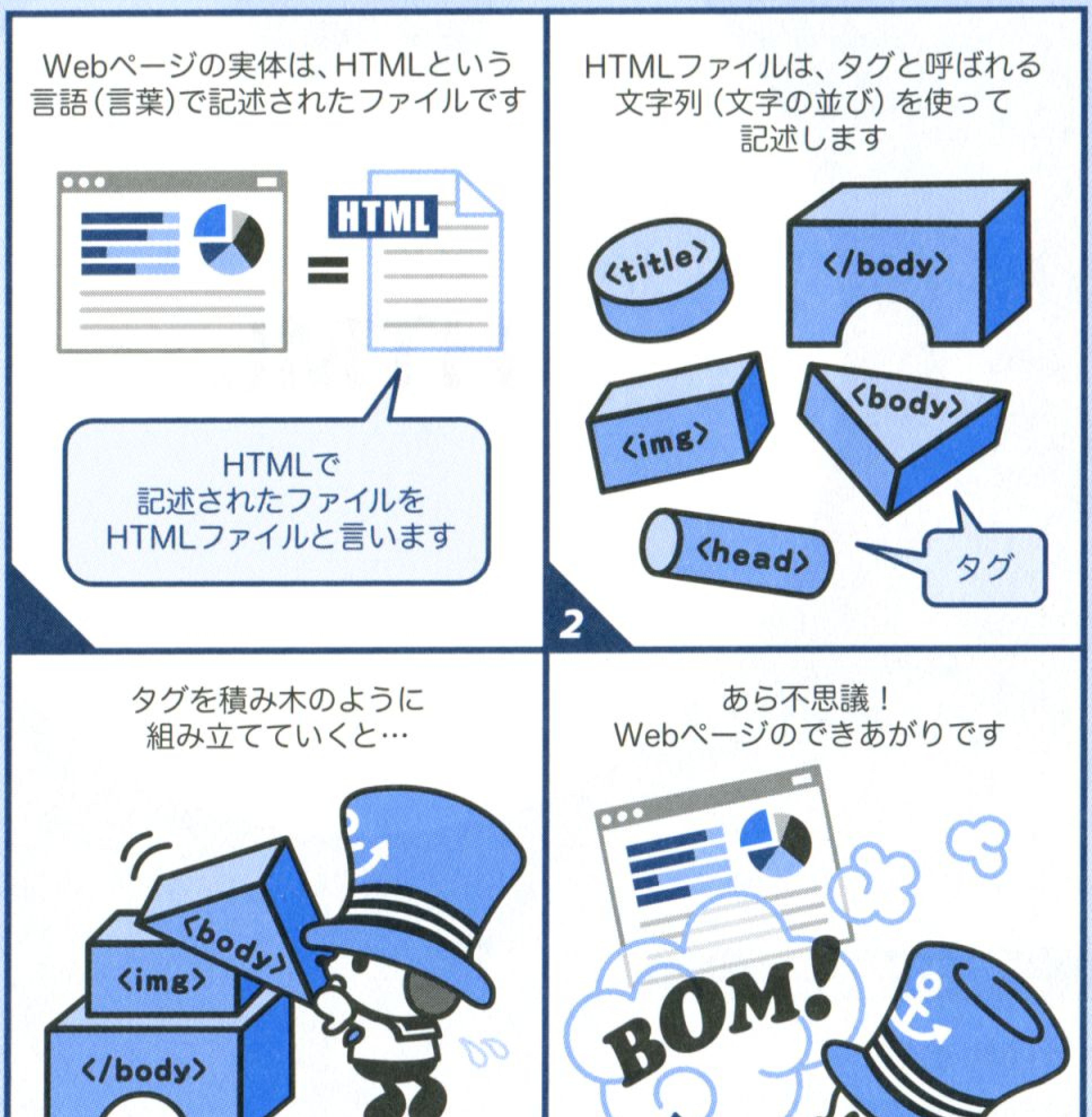
HTML

この章で学ぶこと

- HTMLとは
- HTMLの書き方
- タグの種類と使い方



● Webページは HTMLのタグでできている ●



JavaScriptは、HTMLとCSSで作られた静的なWebページを操作し、動的なWebページを作成するための技術です。JavaScriptの学習に入る前に、まず本章と次章で静的なWebページの作成方法を学びましょう。

2 1 HTMLとは

Webページの実体

ふだん私たちがブラウザ上で見ているWebページは、文字や画像が見やすく配置され、見栄え良く装飾されています。しかし実は、Webページの実体は拡張子「.html」のテキストファイルです。このファイルには文字だけでページの内容が記述されており、この記述に使うのが**HTML**（HyperText Markup Language）という言語です。HTMLで記述されたファイルのことを**HTMLファイル**や**HTML文書**と呼びます。

正式名称のHyperText Markup Languageからわかるように、HTMLは**マークアップ言語**の１つです。マークアップ言語では、タイトルや見出し、段落、箇条書きなど文書が持つ内容を、**タグ**と呼ばれる特殊な**文字列**（文字の並び）で囲む形式で記述します。

HTMLは、Webページに表示する内容をタグを使って記述するマークアップ言語です。

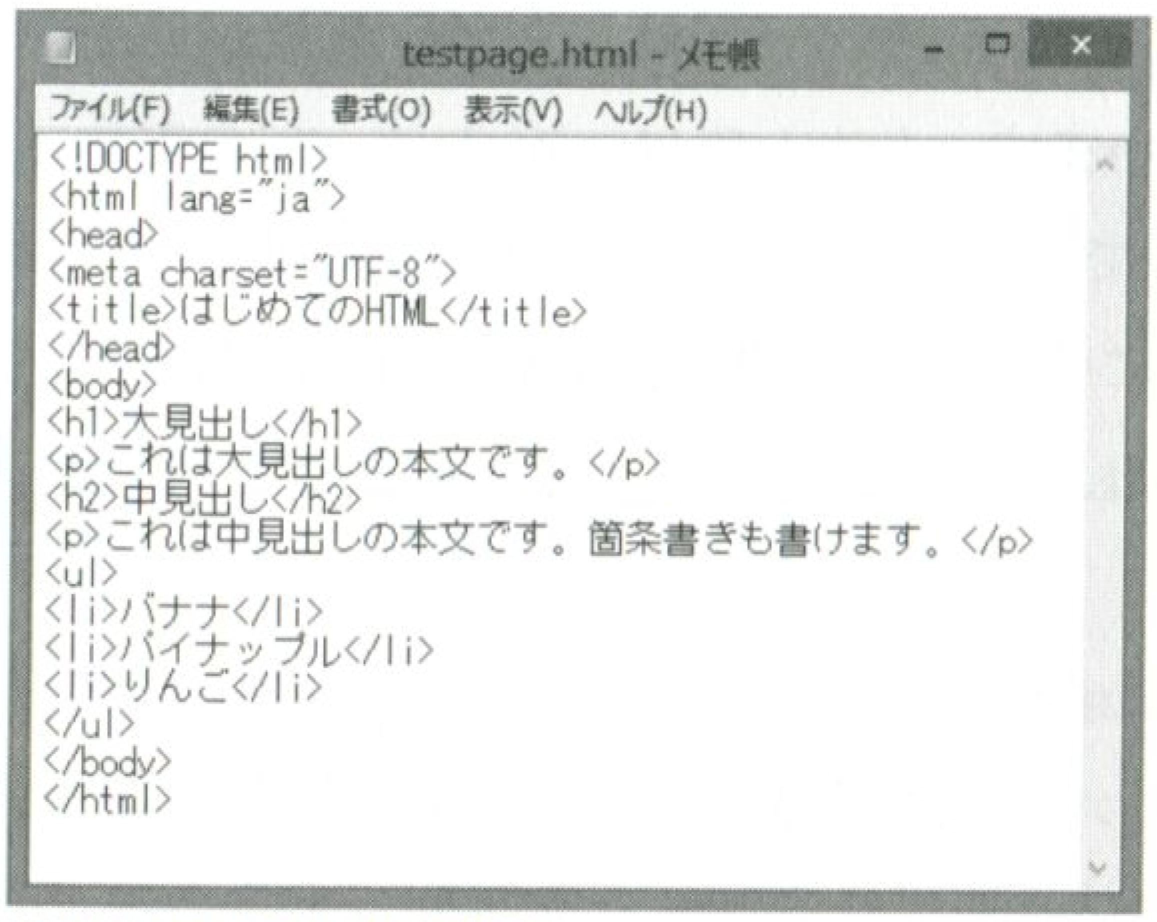
用語解説

【マークアップ言語】

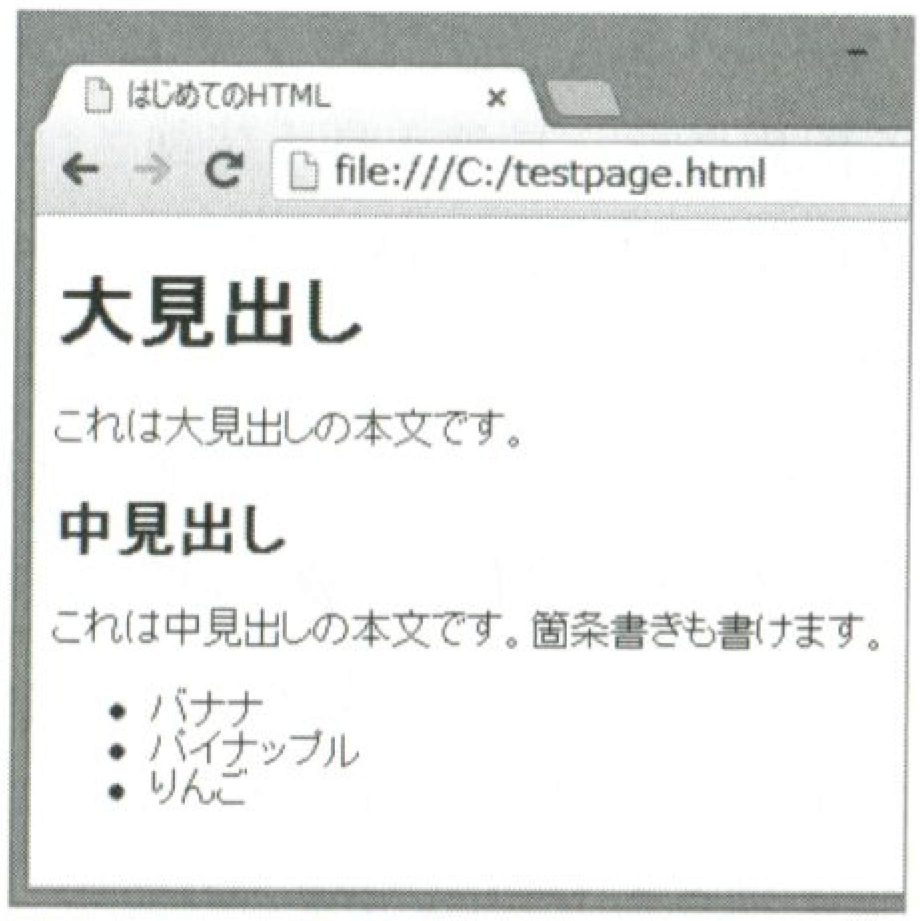
文書やデータの内容（構造）をテキストファイルに記述するための言語です。主なマークアップ言語としてはHTMLやXML（eXtensible Markup Language）があります。マークアップ言語はプログラミング言語ではないことに注目してください。マークアップ言語は「文書やデータの構造を記述（表現）」し、プログラミング言語は「パソコンに対するなんらかの命令を記述」します。

言葉だけだとわかりにくいので、実物を見てみましょう。次ページの2つの画面を見てください。左側がHTMLファイルをテキストエディタ（メモ帳）で開いた画面、右側がそれをブラウザで表示したときの画面です。まずHTML

ファイルの内容ですが、<head>や</ head>など<～>の部分がタグです。日本語の文字がタグで囲まれている（挟まれている）のがわかりますね。タグの意味は後述しますので、いまは気にしないでください。続いて、それをブラウザで表示した画面ですが、だいぶ違って見えるのがわかります。HTMLで記述したタグの意味や構造をブラウザが解釈し、見やすく表示してくれているのです。



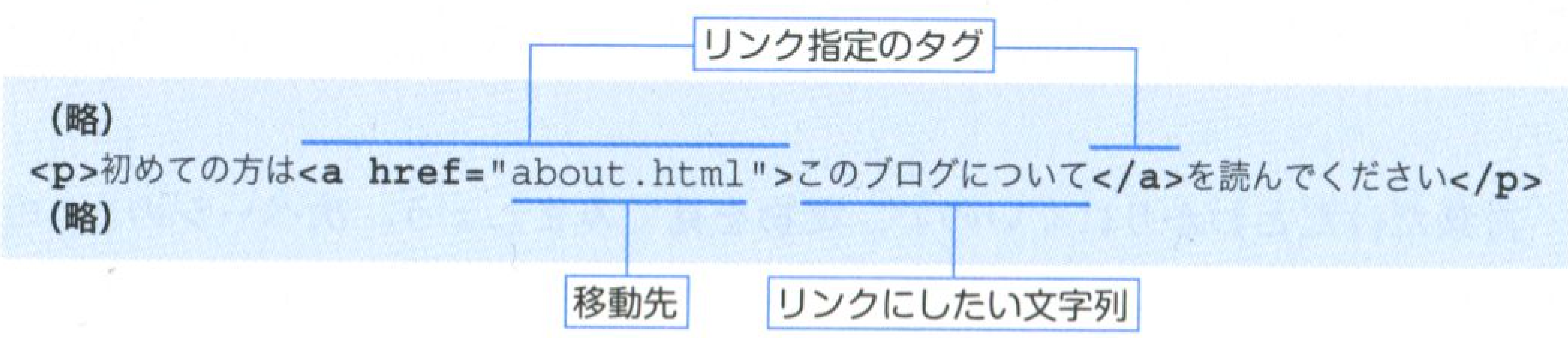
HTMLファイルの記述例

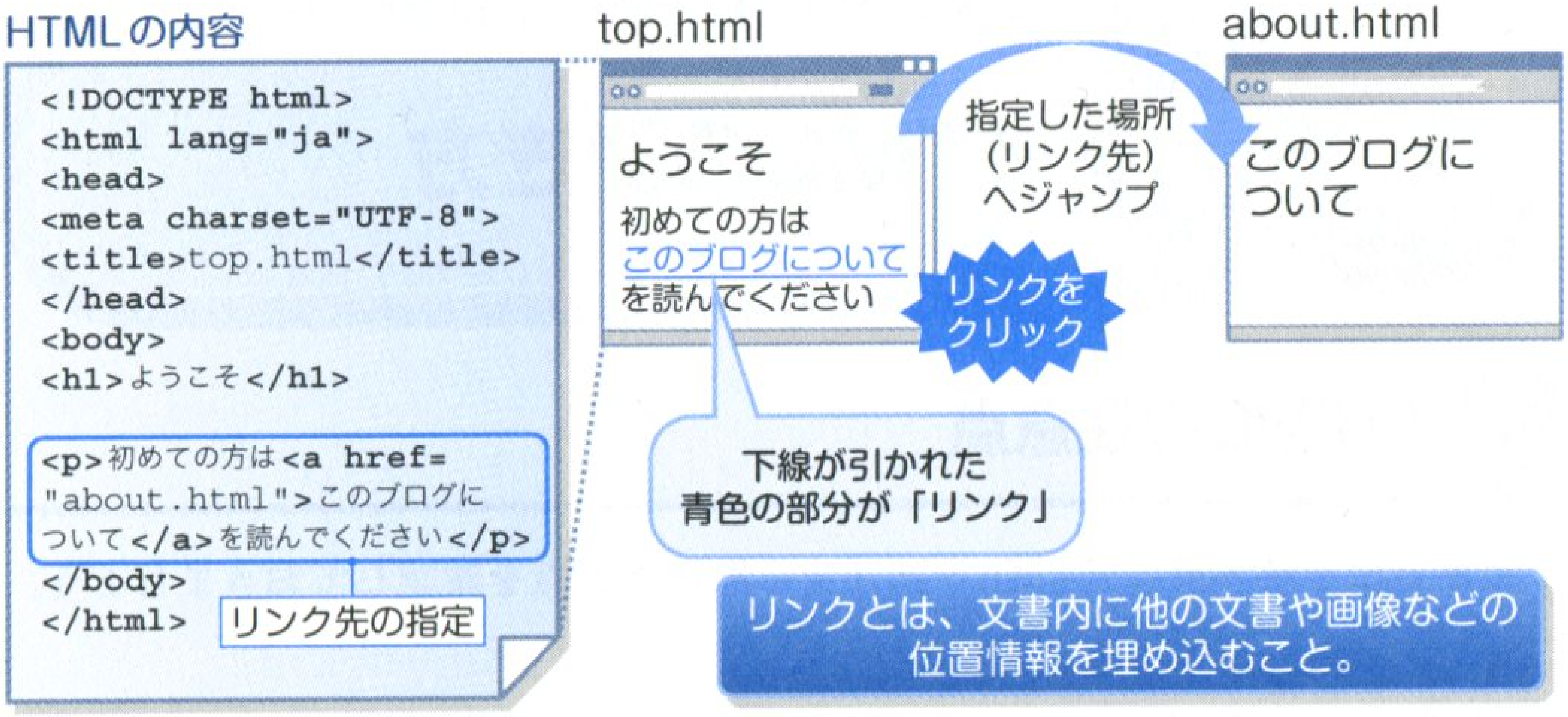


HTMLファイルをブラウザで表示

もともとHTMLは、膨大な量の文書を閲覧しやすくする目的で開発されました。たとえば、文書の中にある専門用語が出てきた場合、その専門用語について解説されている別の文書をすぐに参照できれば便利でしょう。これを可能にしたのがHTMLによる**リンク**です。HTMLにはリンク以外にも、文書を構造化したり、画像を参照する役割などがあります。

たとえば、右ページの図のように、top.htmlの文字をクリックするとabout.htmlへ移動するようにするためにはリンクを使います。リンクにしたい文字列をとというタグで囲むと、Webページ上にリンクが表示されます。





リンクの動作イメージ

なお、HTMLの最新バージョンは5です。2008年にHTML5の仕様についてドラフト版（下書き）が公開され、2014年に正式勧告（正式リリース）の予定です。すでに主要なブラウザのほとんどがHTML5に対応していますので、本書ではこのバージョンについて解説します。

Point!

- Webページの実体は、HTMLという言語で記述されたHTMLファイルである。
- HTMLファイルは複数のタグによって構成される。



■ Check Test 1

Q1 以下はHTMLの説明です。空欄AとBを適切な語で埋めてください。

・HTMLは［ A ］を記述するためのマークアップ言語。［ A ］が持つ内容を、［ B ］と呼ばれる特殊な文字列で囲む形式で記述する

解答は巻末に掲載

2 HTMLの書き方

HTMLの注意点

記述方法を学ぶ前にHTMLを記述する際の注意点を確認しておきましょう。

◆注意点1 HTMLのタグは、必ず半角英数字で記述する

○	<head>	すべて半角
×	＜ｈｅａｄ＞、＜head＞、＜ｈead＞	すべて全角または一部全角
×	< head>、<h ead>、<he ad>	タグのどこかにスペースがある

タグに全角文字を使ったり、タグのどこかにスペース（空白）を入れてしまうと、ブラウザがタグとして認識できなくなります（ただし、次項で解説するタグの属性だけは例外で、属性の前に必ず半角スペースが必要です）。

◆注意点2 HTMLは大文字と小文字を区別しない

○	<head>	すべて小文字
○	<HEAD>、<Head>、<HeAD>	すべて大文字または一部大文字

HTML5は、英字の大文字と小文字を区別しないため、どちらで記述してもかまいません。ただし、一般的には小文字を利用することが多く、特定の意味があって大文字にするもの以外は小文字で統一することをおすすめします。

基本的な書き方

では、HTMLの記述方法を見ていきましょう。HTMLのタグにはさまざまな種類がありますが、記述方法はどれも同じで、Webページに表示するコンテンツをタグと呼ばれる特殊な文字列で囲みます。

文法 タグ／要素の記述

```
開始タグ        終了タグ
<タグ名>内容</タグ名>
      要素
```

■記述例（段落を表わす<p>要素）

```
<p>こんにちは</p>
```

「タグ名」にはタグの名称、「内容」にはWebページに表示するコンテンツが入ります。**開始タグ**と**終了タグ**でコンテンツを囲み、終了タグはタグ名の前にスラッシュ（/）を記述します。また、開始タグから終了タグまでの全体を**要素**と呼びます。上の記述例でいうと、開始タグは<p>、終了タグは</p>なので、<p>から</p>までが1つの要素（1段落）となります。

◇ 空要素

要素の種類によっては、終了タグが存在しないものもあります。そのような要素は**空要素**と呼びます。

文法 空要素の記述

```
<タグ名>
```

■記述例（改行を表わす
要素）

```
<br>
```

◇ 属性

各種要素は、それぞれ異なる**属性**を持っています。属性とは、要素に付ける付加情報のことで、仮にリンク要素であればリンク先のURLなどを指定します。また、次章で学ぶCSSを一部の要素にだけ適用する場合や、JavaScriptで特定の要素を操作する場合などにも、あらかじめ属性を付けておきます。属性は、開始タグに記述します。1つの要素に対して複数の種類の属性を付けることが可能です。

文法 属性の記述

開始タグ　　終了タグ

<タグ名 属性="値">内容</タグ名>

半角スペース

<タグ名 属性1="値" 属性2="値">内容</タグ名>

半角スペース

■記述例（リンクを指定する<a>要素）

```
<a href="test.html">こんにちは</a>
```

属性の前には必ず半角スペースが必要です。また、属性値はダブルクォート（"）の代わりにシングルクォート（'）で囲んでもかまいません。たとえば、上記の記述例でいえば、開始タグは<a href='test.html'>と記述してもかまいません。また、属性の指定順序に決まりはありません。

HTMLの構成

HTML文書は、いくつもの要素を組み合わせて構成します。記述する際は、要素の中に別の要素を入れ込んでいく構造にします。この構造を**入れ子**または**ネスト**と呼びます。

入れ子（どちらも同じ入れ子）

```
<body>
 <p>
  <a>
  </a>
 </p>
</body>
```

```
<body><p><a></a></p></body>
```

このときに注意しなければならないのは、終了タグの位置です。

良い例

```
<div><p>こんにちは</p></div>
```

悪い例

```
<div><p>こんにちは</div></p>
```

タグが交差するように配置してはいけません。必ず要素の外側を包むように別の要素を配置します。

Point!

- Webページのコンテンツを開始タグと終了タグで囲む。
- 開始タグから終了タグまでの全体を要素と呼ぶ。
- 空要素という終了タグがない要素もある。
- 要素には属性という付加情報を指定できる。属性の前には半角スペースが必要。
- 複数のタグを組み合わせて使う場合、別のタグと交差しないよう必ず入れ子で記述する。



■ Check Test 2

Q1 以下はタグの説明です。空欄A～Cを適切な語で埋めてください。

・タグは［ A ］と［ B ］の２種類の記号で構成される文字列。開始タグから終了タグまでの全体を［ C ］と呼ぶ

Q2 ネストとはなんですか？

➲解答は巻末に掲載

2-3 HTMLの作成と表示

簡単なHTMLを作ろう！

それでは、実際にHTMLファイルを作成し、ブラウザで表示してみましょう。HTMLやJavaScriptのようなクライアントサイドの技術しか利用していない場合、実際にインターネット上にページを公開しなくても、ローカルPC（個人のパソコン）上で実行結果を確認できます。次の手順に従って実行してみてください。

◆ 手順1 HTMLの記述

テキストエディタにリスト2.1のソースコードを入力します。リスト内の␣は、半角スペースを表わします。

リスト2.1 最低限のタグで構成されたHTML（list2_1.html）

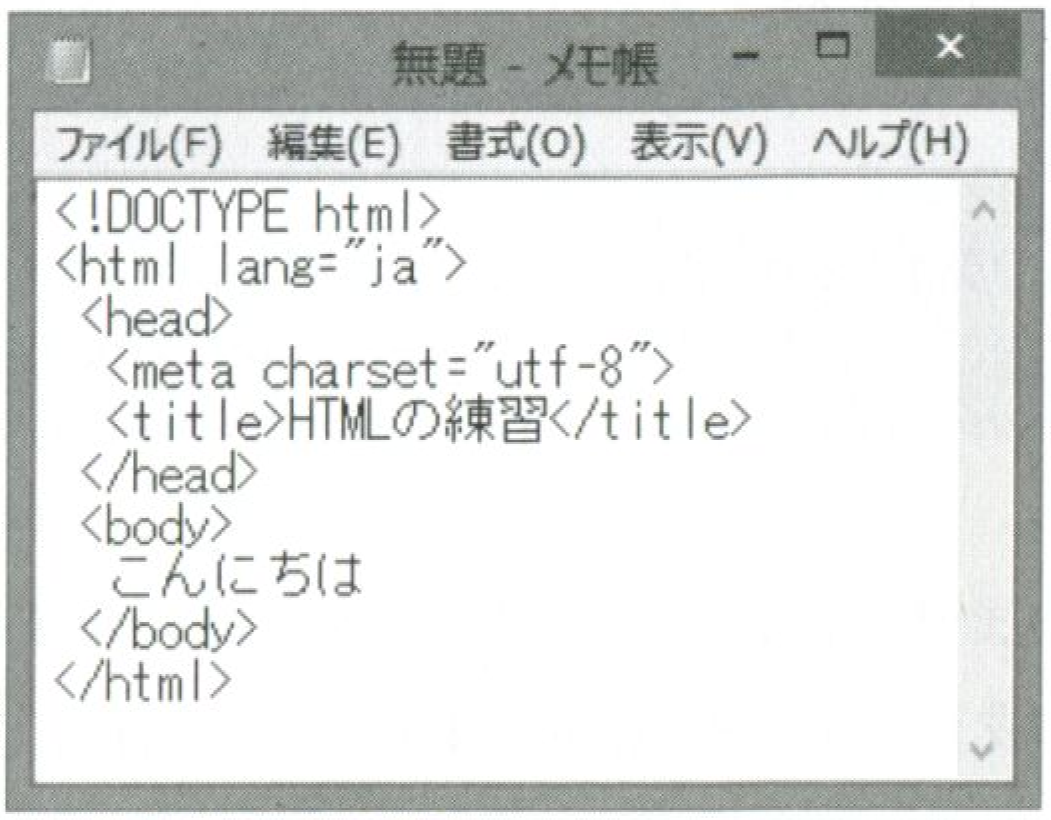
```
<!DOCTYPE html>
<html lang="ja">
␣<head>
␣␣<meta charset="utf-8">
␣␣<title>HTMLの練習</title>
␣</head>
␣<body>
␣␣こんにちは
␣</body>
</html>
```

HTMLを見やすくするため、半角スペースを入力して字下げ

用語解説 ▶【ソースコード】

ソースは「元となるもの」、コードは「命令文」を意味します。ここでは、「Webページの元となっている命令文」という意味で使っています。「ソース」と「コード」はそれぞれ単独で使う場合もあります。コード、プログラム、スクリプトは同義語として使われます。

ちなみに、以下はメモ帳で入力を終えた状態です。タグはすべて半角文字で入力し、アルファベットのつづりに間違いがないか注意してください。

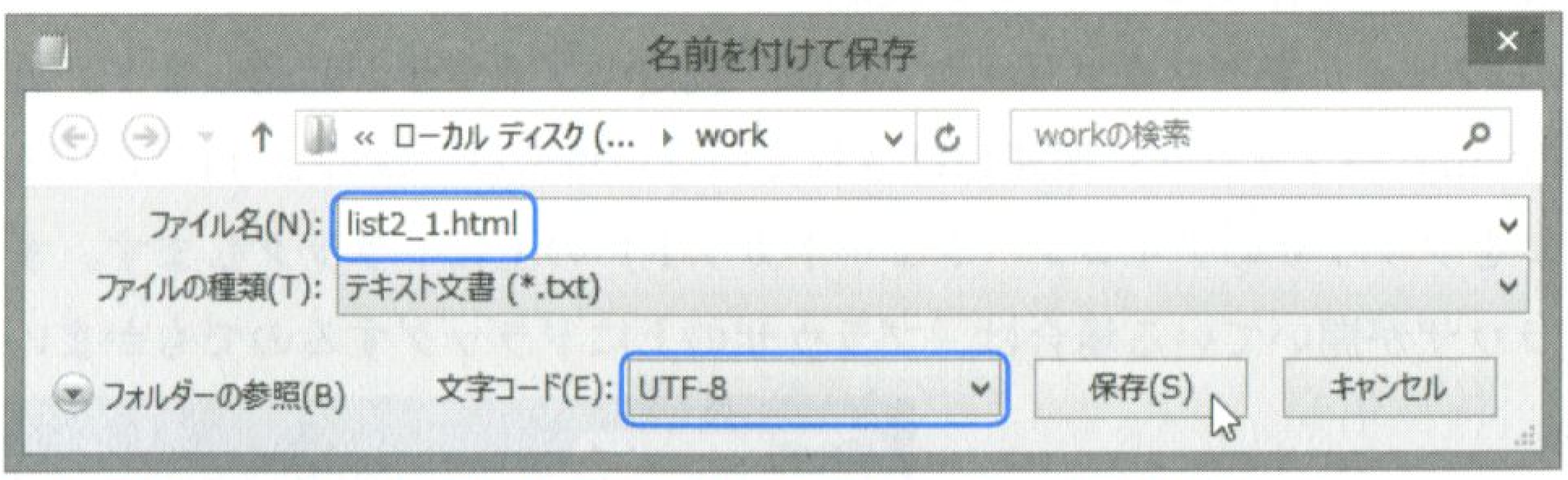


メモ帳でHTMLを入力

> **Note** コードを見やすくするための字下げを**インデント**と呼びます。インデントは、Webページの表示に影響しません。インデントは、タブまたは半角スペースで行なってください。

◆ 手順2 ファイルの保存

HTMLファイルとして保存します。メモ帳の場合は、[ファイル(F)] メニューから [名前を付けて保存(A)…] をクリックし、「名前を付けて保存」ダイアログで保存します。保存する際の注意点として、テキストエディタでファイルの**文字コード**を「UTF-8」にする必要があります。ファイル名は「list2_1.html」と入力します。



「名前を付けて保存」ダイアログでファイル名入力と文字コード選択を行なう

用語解説

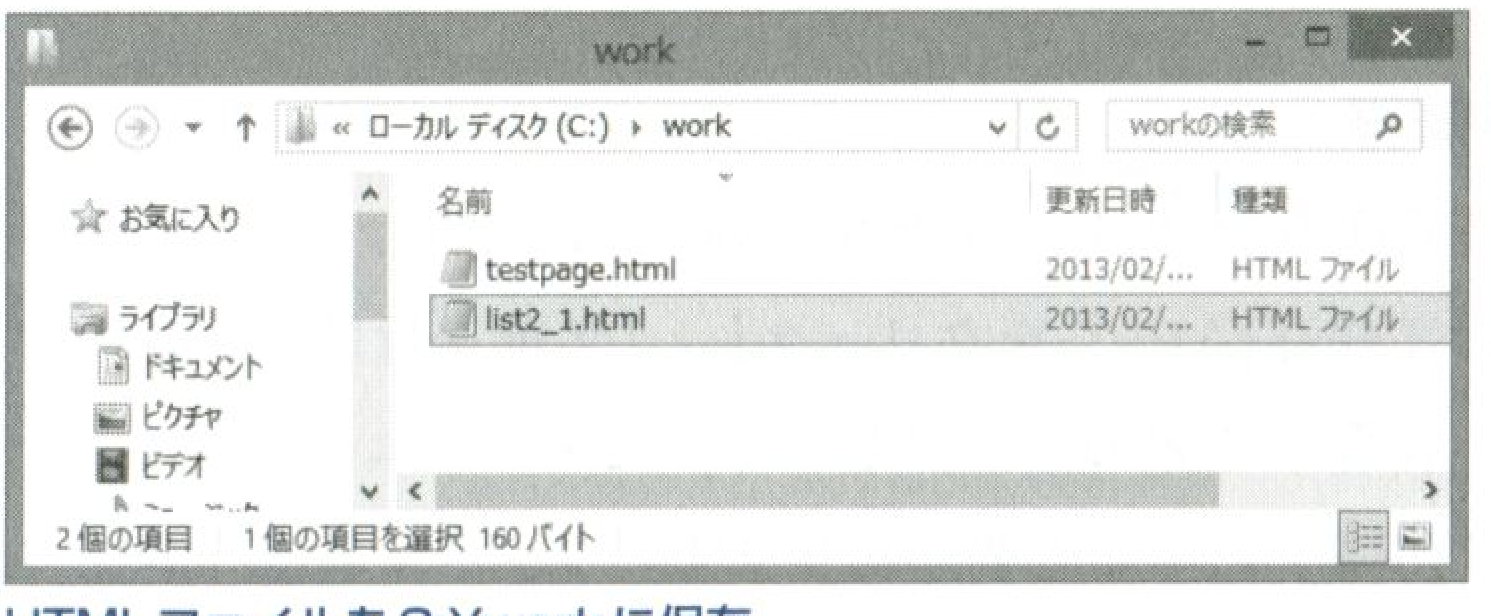
【文字コード】

パソコンで利用される文字は、内部的には番号で管理されています。文字と番号の対応関係のことを文字コードと呼びます。世界各国の文字を表現するためにさまざまな種類の文字コードが開発されていて、よく使われる文字コードに「UTF-8」や「Shift_JIS」、「EUC-JP」などがあります。

注意 HTMLの文字コードとファイル名

HTML5では、文字コードにUTF-8を使うことが推奨されています。HTMLファイルを保存するときには、文字コードを必ずUTF-8に指定するのを忘れないようにしましょう。UTF-8以外の文字コードで保存すると「文字化け」の原因となるため、注意してください。
また、HTMLのファイル名には、半角の英数字、ハイフン（-）、アンダースコア（_）のみの使用が推奨されます。

ファイルの保存場所はどこでもかまいませんが、ここではCドライブの直下にあらかじめworkというフォルダを作成し、そこへ保存しましょう。［保存］ボタンをクリックすると、ファイルが保存されます。



HTMLファイルをC:¥workに保存

◆手順3 ファイルをブラウザで表示

HTMLファイルをダブルクリックすれば、既定のブラウザが起動します。拡張子「.html」がブラウザに関連付けされていない場合は、次のようにHTMLファイルをブラウザのアイコン（ショートカット）の上にドラッグします。すでにブラウザが開いている場合は、ブラウザの上にドラッグするのでもかまいません。



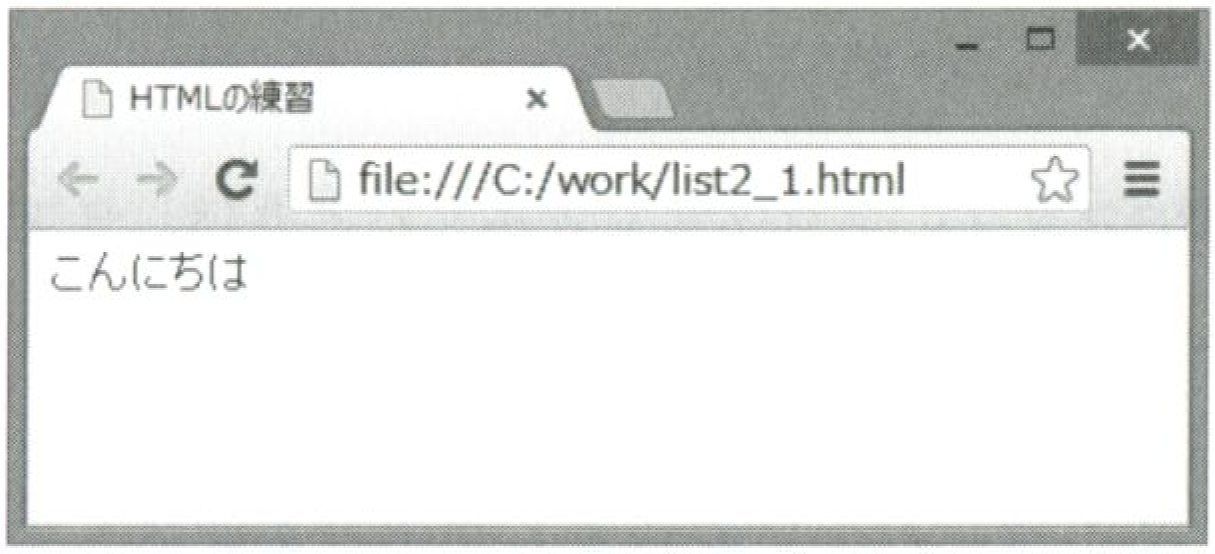
ファイルをブラウザのアイコンにドラッグ



ファイルをブラウザ上にドラッグ

ブラウザにHTMLファイルが表示されたでしょうか？　Chromeでの表示結果は次のとおりです。

■実行結果



タグの意味

では、記述した各タグの意味について確認しましょう。

```
<!DOCTYPE html>  ――――――――――――――― ❶
<html lang="ja">  ―――――――――――――― ❷
 <head>  ―――――――――――――――――――― ❸
  <meta charset="utf-8">  ――――――――― ❹
  <title>HTMLの練習</title>  ――――――― ❺
 </head>
 <body>  ―――――――――――――――――――― ❻
  こんにちは
 </body>
</html>
```

❶ <!DOCTYPE html>

HTML5で記述された文書であることを表わすタグ。終了タグはありません。

❷ <html lang="ja">

HTML文書であることを表わす<html>タグ。文書全体をこのタグで囲みます。lang属性は使用する自然言語（日本語や英語など）を指定します。日本語の場合、属性値は「ja」です。

❸ <head>

文書全体に関する情報を定義する<head>タグ。このタグ自体はあまり意味を持たず、中に入れ子にして記述するタグがさまざまな意味を持ちます。

❹<meta charset="utf8">

文書の文字コードを指定する**<meta>**タグ。ここで指定した文字コードとHTMLファイルを保存するときに指定した文字コードが異なる場合、文字化けが発生してしまうので気をつけましょう。

❺<title>

文書のタイトルを定義する**<title>**タグ。ブラウザで表示したときに、タブに表示される文字列となります。

❻<body>

本文を表わす**<body>**タグ。Webページ上に表示する文字列を記述します。

これらは、HTMLの最低限のタグです。<body>要素の中に文書（Webページ）の本文を記述していきます。この要素の中に入れ子でさまざまなタグを記述できますが、これについては次節で見ていきます。

Point!

- ▶ コードを見やすくするためにインデント（字下げ）を使う。
- ▶ HTMLファイルは文字コードをUTF-8にして保存する。



■ Check Test 3

Q1 HTML文書全体を囲むタグはなんですか？

Q2 文字コードを指定するために<meta>タグに付ける属性はなんですか？

➲解答は巻末に掲載

2 4 主要なタグ

<body>要素の中に記述する要素

前節では、HTMLを構成する最低限のタグを紹介しました。この他にもHTMLにはさまざまなタグが用意されていますが、それらのほとんどが文書の本文を表わす<body>要素の中に記述するタグです。非常に多くのタグと属性があるため、主要なものを以下にまとめます。すべてをおぼえる必要はありませんが、その使い方と組み合わせ方をざっと把握しておきましょう。

◇ 終了タグのあるタグ

<table>
<tr><th>タグ名</th><th colspan="2">説明／属性</th></tr>
<tr><td rowspan="2">h1</td><td colspan="2">見出しを定義する。h1 ～ h6 まであり、h1 が最も高レベルで、h6 が最も低レベルな見出し</td></tr>
<tr><td colspan="2">例 <h1>見出し</h1></td></tr>
<tr><td rowspan="2">p</td><td colspan="2">文章の段落を定義する</td></tr>
<tr><td colspan="2">例 <p>
文章の段落を
定義します。
</p></td></tr>
<tr><td rowspan="2">div</td><td colspan="2">特に意味を持たないタグ。複数のタグをまとめて扱うときに使う</td></tr>
<tr><td colspan="2">例 <div>
<h1>見出し</h1>
<p>段落</p>
</div></td></tr>
<tr><td rowspan="5">a</td><td colspan="2">リンクを定義する
▼<a> タグが持つ属性</td></tr>
<tr><th>属性名</th><th>指定内容</th></tr>
<tr><td>href</td><td>リンク先の URL を指定する</td></tr>
<tr><td>target</td><td>リンク先をどのように表示するかを指定する。指定値は以下
_blank　新しくタブを開く
_self（既定値）　現在のタブに表示する</td></tr>
<tr><td colspan="2">例 TOPへ</td></tr>
<tr><td rowspan="2">button</td><td colspan="2">ボタンを定義する</td></tr>
<tr><td colspan="2">例 <button>ボタン</button></td></tr>
</table>

◆ 終了タグのないタグ（空要素）

<table>
<tr><th>タグ名</th><th colspan="2">説明／属性</th></tr>
<tr><td rowspan="7">img</td><td colspan="2">画像を参照する
▼<img>タグが持つ属性</td></tr>
<tr><th>属性名</th><th>指定内容</th></tr>
<tr><td>src</td><td>画像の参照先を指定する</td></tr>
<tr><td>alt</td><td>画像がなんらかの理由で表示できなかった場合に、画像の代わりに表示する文字列を指定する</td></tr>
<tr><td>width</td><td>画像の幅を指定する。指定できる単位は CSS の width と同じ（60 ページ）</td></tr>
<tr><td>height</td><td>画像の高さを指定する。指定できる単位は CSS の height と同じ（60 ページ）</td></tr>
<tr><td colspan="2">例 <img src="flower.jpg" alt="花" width="300px" height="200px"></td></tr>
<tr><td rowspan="2">br</td><td colspan="2">文章内で改行する</td></tr>
<tr><td colspan="2">例
タグ</td></tr>
<tr><td rowspan="2">!--
--</td><td colspan="2">コメント（説明文）を記述する</td></tr>
<tr><td colspan="2">例 <!-- 説明文 --></td></tr>
</table>

◆ すべてのタグに付けられる属性

<table>
<tr><th>属性名</th><th>説明</th></tr>
<tr><td rowspan="2">id</td><td>要素を識別するための ID。文書内で重複する値を指定することはできない</td></tr>
<tr><td>例 <div id="header">…</div></td></tr>
<tr><td rowspan="2">class</td><td>CSS のクラス名を指定する（63 ページ）</td></tr>
<tr><td>例 <div class="container">…</div></td></tr>
</table>

以上、<body>要素の中に記述する主要なタグについてまとめました。続いて、これらの中でも特に頻繁に使われるリンクと画像の表示について、実例を見ながら詳しい使い方を学びましょう。

リンクの利用

リンクは、あるWebページから別のWebページへ移動する機能です。

文法 リンクの記述

```
<a href="リンク先URL">リンク文字列</a>
```

「リンク先URL」の指定方法には、**絶対パス**指定と**相対パス**指定の2通りがあります。



絶対パスと相対パスのイメージ

◆ 絶対パス指定

絶対パスはフルパスとも呼びます。「http://」から始まるURLをすべて記述する方法です。自分のWebサイトから別のWebサイトへ移動する場合は、この指定方法を使います。

リスト2.2　リンクの表示（絶対パス指定）

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>HTMLの練習</title>
 </head>
 <body>
  <a href="http://www.google.co.jp/">グーグルへ移動</a>
 </body>
</html>
```

■実行結果



◆ 相対パス指定

現在のWebページから見た、対象ページまでの位置を指定する方法です。自分のWebサイト内の別のWebページへ移動する際に使用します。相対パスを指定する場合、記述方法に次のルールがあります。

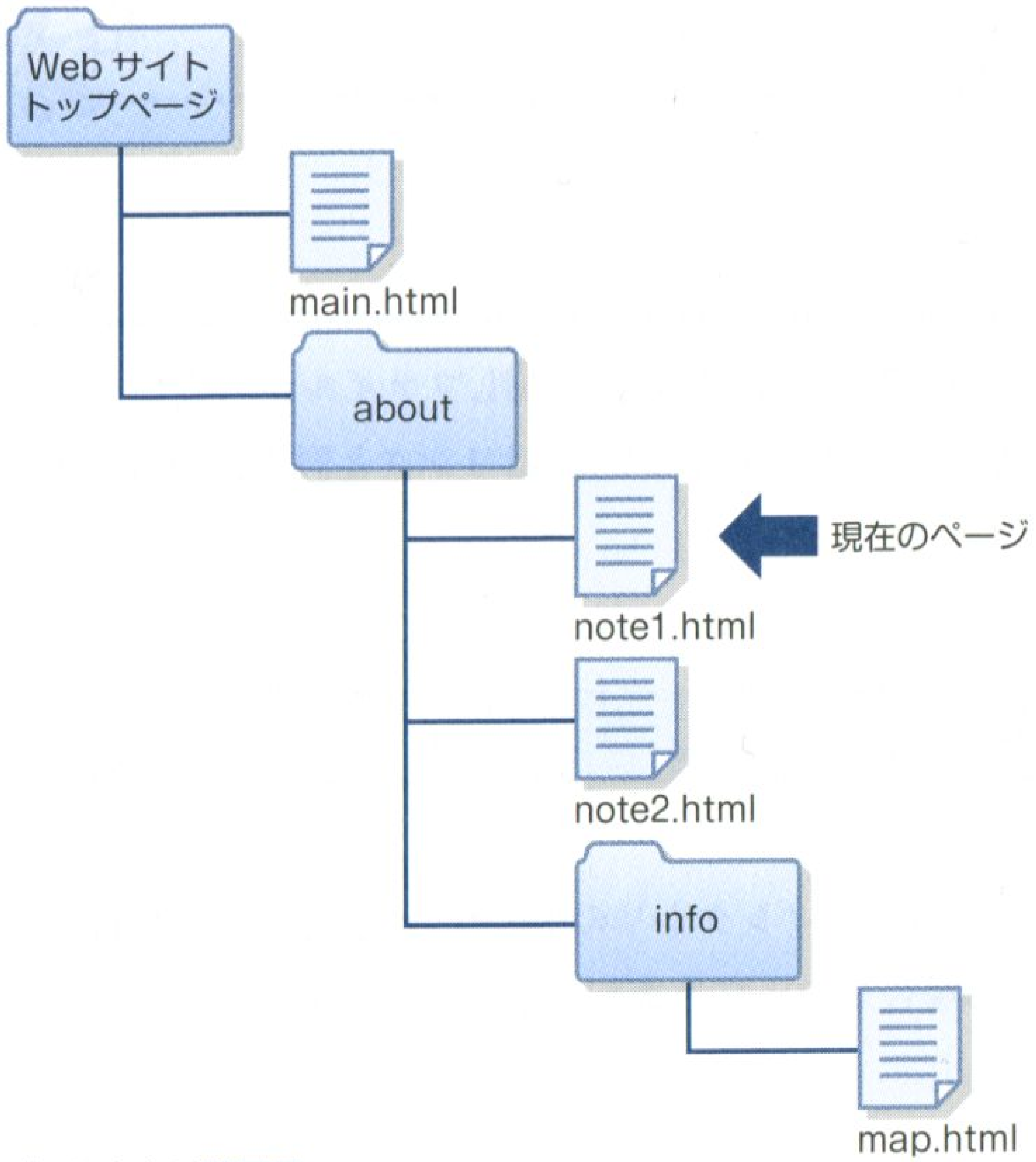
ルール1 ▶ 同一ディレクトリ内のWebページに移動する場合はファイル名の指定だけで良い。

ルール2 ▶ ディレクトリの中を表わすには、ディレクトリ名の後ろにスラッシュを付ける。

ルール3 ▶ 1つ上のディレクトリは .. という記号で表わす。

用語解説 ▶ **【パス、ディレクトリ】**
パスはファイルが配置されている場所、**ディレクトリ**はサーバー上に存在するフォルダのことです。

具体例を見てみましょう。たとえば、次のようなディレクトリ構成のWebサイトがあるとします。



ディレクトリ階層図

この場合、次のようにリンク先URLを指定できます。

リンク先	URL（現在のWebページからの相対パス指定）
note2.html	note2.html
main.html	../main.html
map.html	info/map.html

相対パス指定は、画像ファイルを指定する際にもよく使います。次項で説明する画像の表示の記述例で相対パス指定を使用していますので、記述例はそちらを参照してください。

Point!

- 絶対パスは、Webページから他のWebサイトへ移動するときに使う。
- 相対パスは、自分のWebサイト内でのページ移動や画像の指定などに使う。



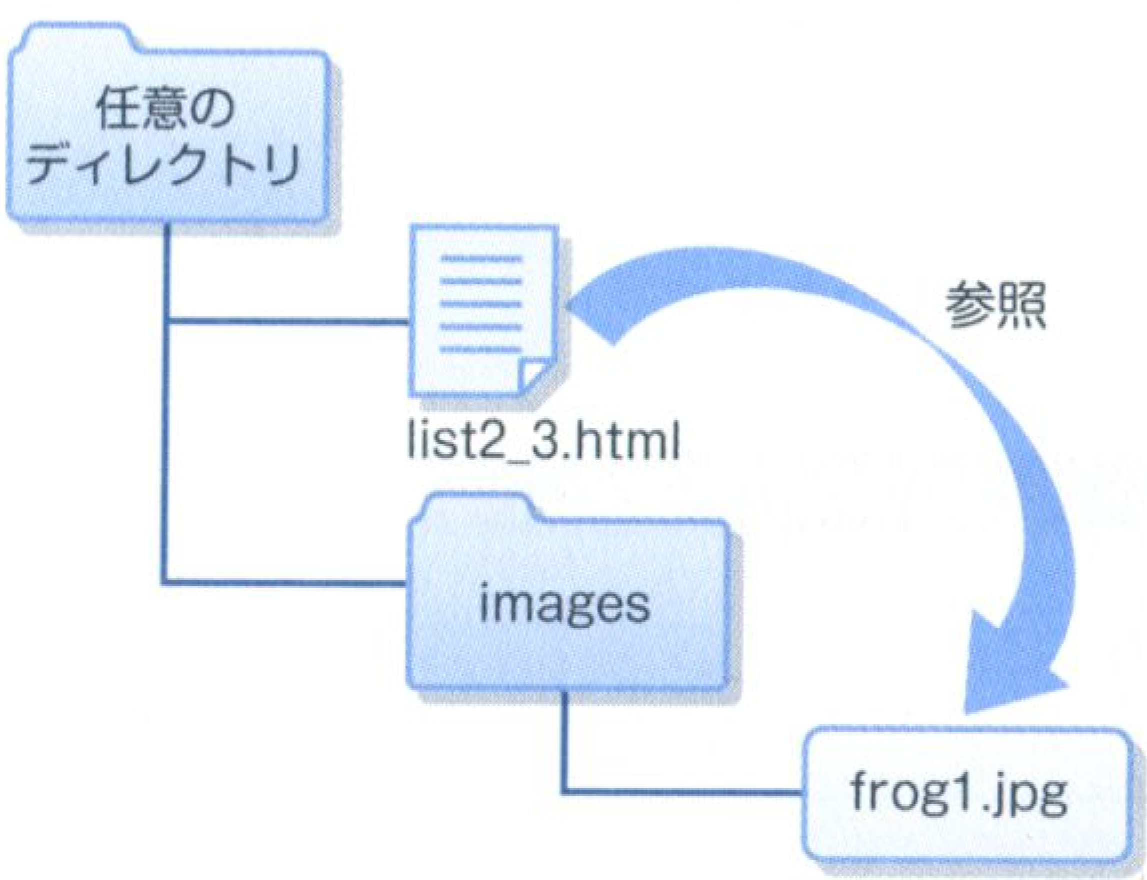
画像の表示

Webページ上に画像を表示するためには、HTMLファイルから画像ファイルを参照するようタグで指定します。画像のURLはリンクと同様、絶対パスまたは相対パスで指定します。

文法 リンクの記述

```
<img src="画像のURL">
```

例として、次のようにimagesディレクトリの中のfrog1.jpgという写真データをWebページに表示する場合を考えてみましょう。



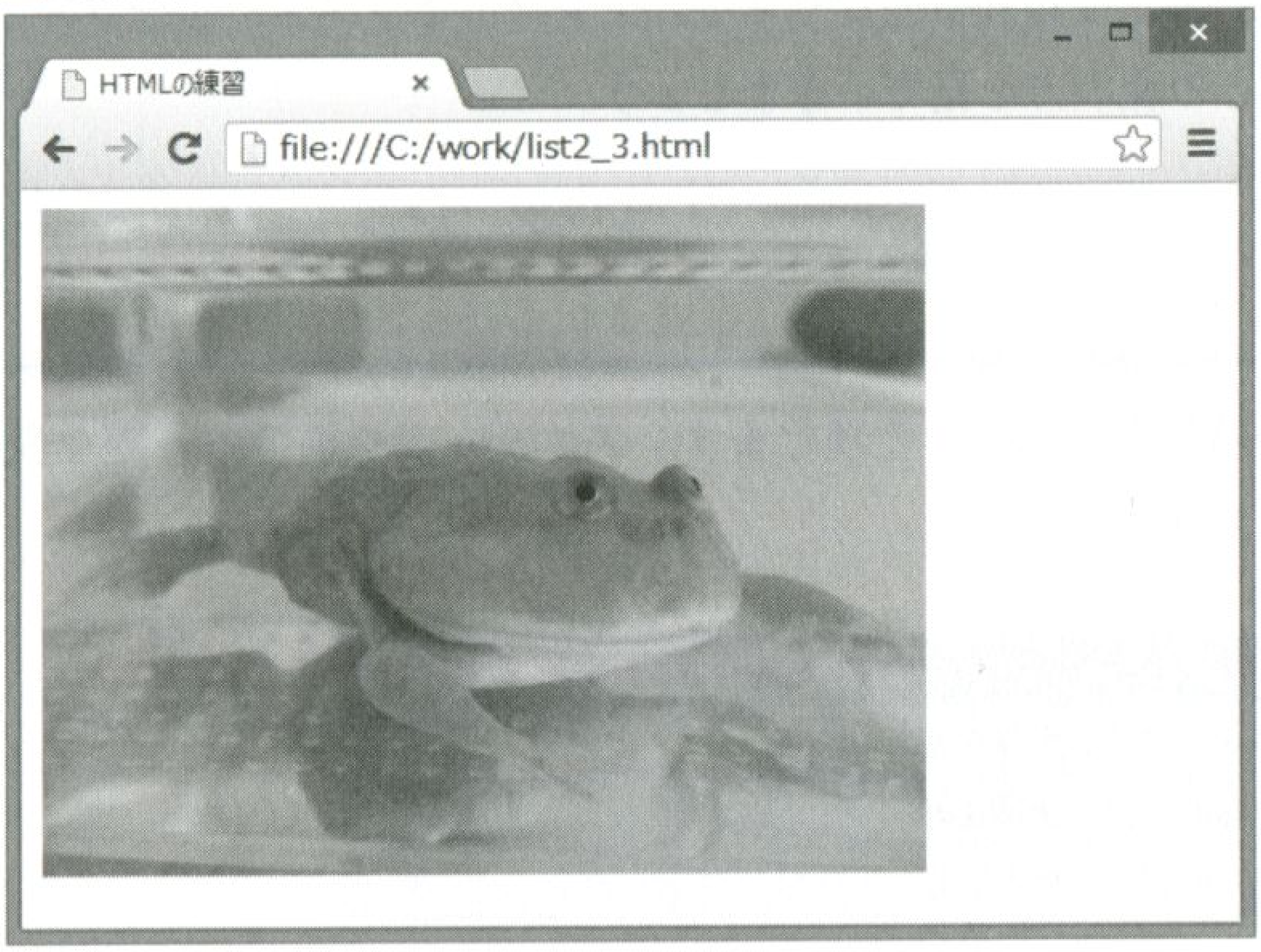
list2_3.htmlからimagesディレクトリの写真データを参照する

この場合、リスト2.3のような相対パス指定を行なう必要があります。

リスト2.3 画像の表示（相対パス指定）

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>HTMLの練習</title>
 </head>
 <body>
  <img src="images/frog1.jpg">
 </body>
</html>
```

■実行結果



注意 **ファイルやディレクトリの名前**

HTML/CSS/JavaScriptで扱うファイル（画像やCSS/JavaScriptファイルなど）やディレクトリの名前には、半角の英数字、ハイフン（-）、アンダースコア（_）のみの使用が推奨されます。

■ Check Test 4

Q1 リンクを定義するタグの名称はなんですか？

Q2 <img>タグに付ける画像の参照先を表わす属性の名称はなんですか？

⮌解答は巻末に掲載

2-5 特殊なタグ

複数組み合わせて１つの部品を生成するタグ

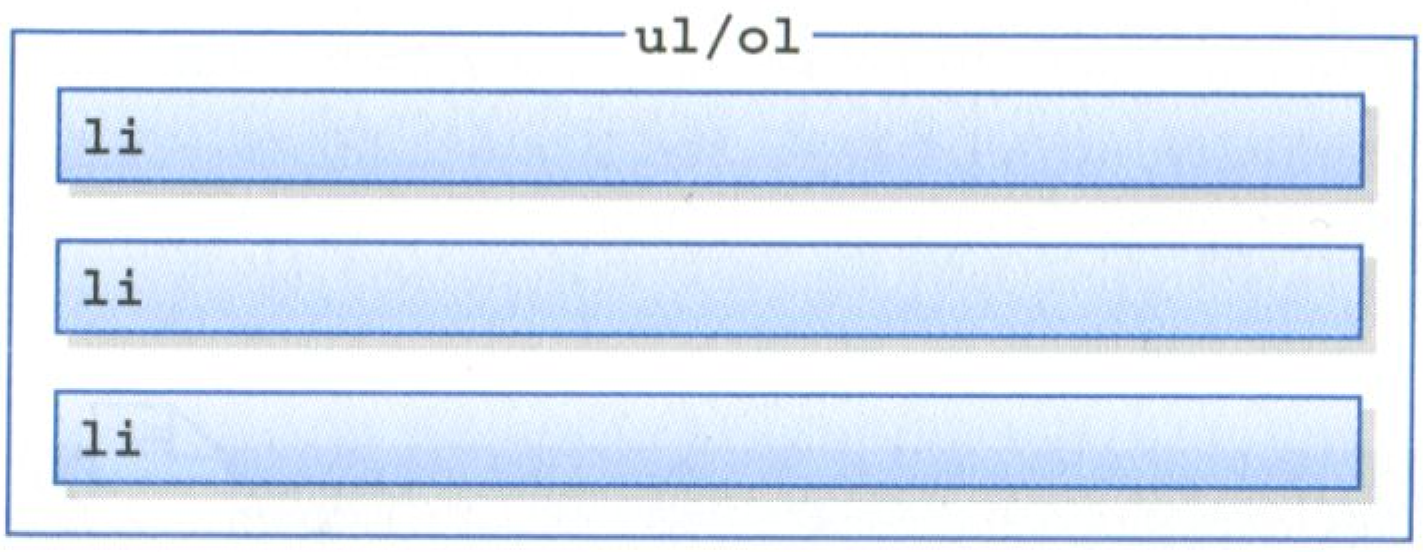
入力欄やボタンなど、Webページの一部分を形成するHTML要素を**部品**と呼びます。複数のタグを組み合わせることで、1つの部品を表わす特殊なタグがいくつかあります。よく使われるのがリスト、セレクトボックス、テーブル、フォームです。これらの使い方を見ていきましょう。

リスト

リストは、複数の項目をまとめて管理します。メニュー表や画像の一覧を表示する際などに利用されます。

タグ名	説明
ul	順序のないリスト全体を囲む
ol	順序のあるリスト全体を囲む
li	リスト内の項目を定義する

これらのタグを使って、次のようなリストを表示してみましょう。



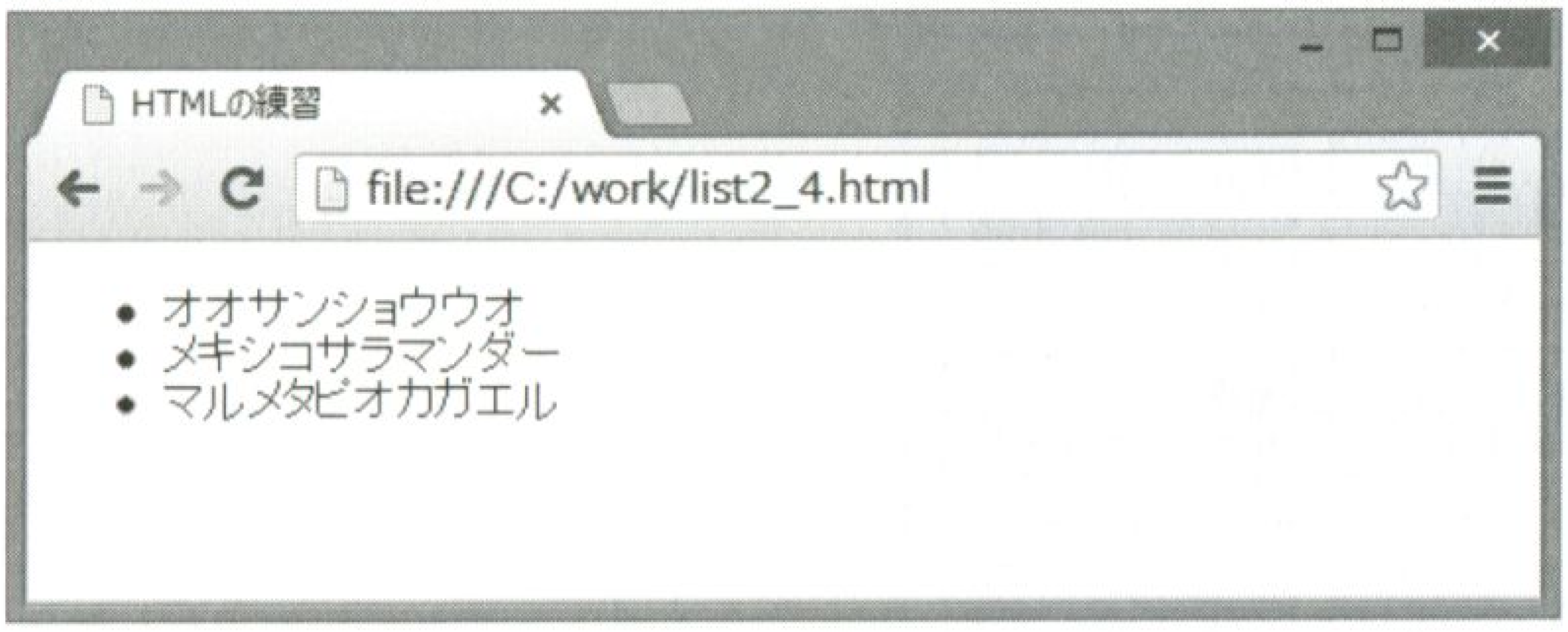
リストの構成イメージ

リスト2.4では、3件の項目を持つリストを順序なしで表示しています。

リスト2.4　順序なしリストの表示

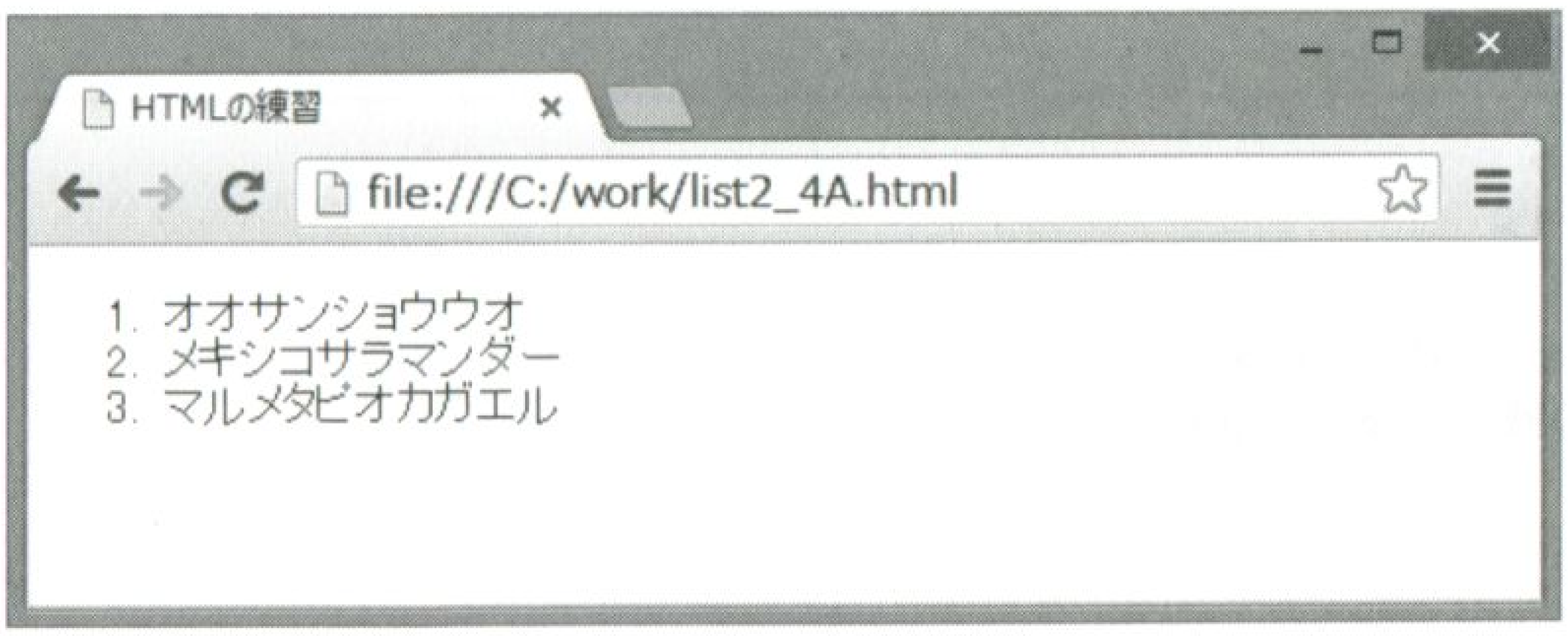
```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>HTMLの練習</title>
 </head>
 <body>
  <ul>
   <li>オオサンショウウオ</li>
   <li>メキシコサラマンダー</li>
   <li>マルメタピオカガエル</li>
  </ul>
 </body>
</html>
```

■実行結果　順序なし



リスト2.4の
を、
（順序ありリスト）に変更すると、次のようになります。

■実行結果　順序あり

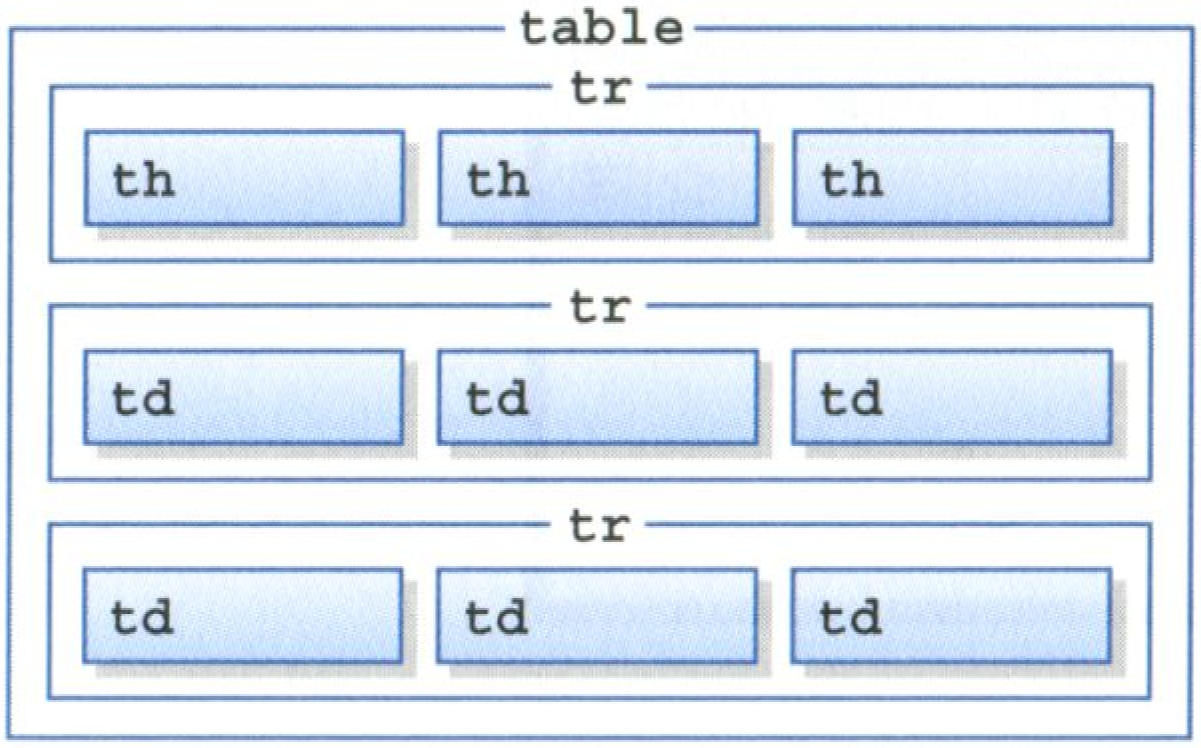


テーブル

テーブルは、データを表形式で管理します。WebページにExcelで作成したような表を表示する場合に利用します。

タグ名	説明
table	テーブル全体を囲む
tr	テーブルの列を定義する
th	テーブルの見出しセルを定義する
td	テーブルのデータセルを定義する

このタグを使って、次のような構成のテーブルを表示してみましょう。



テーブルの構成イメージ

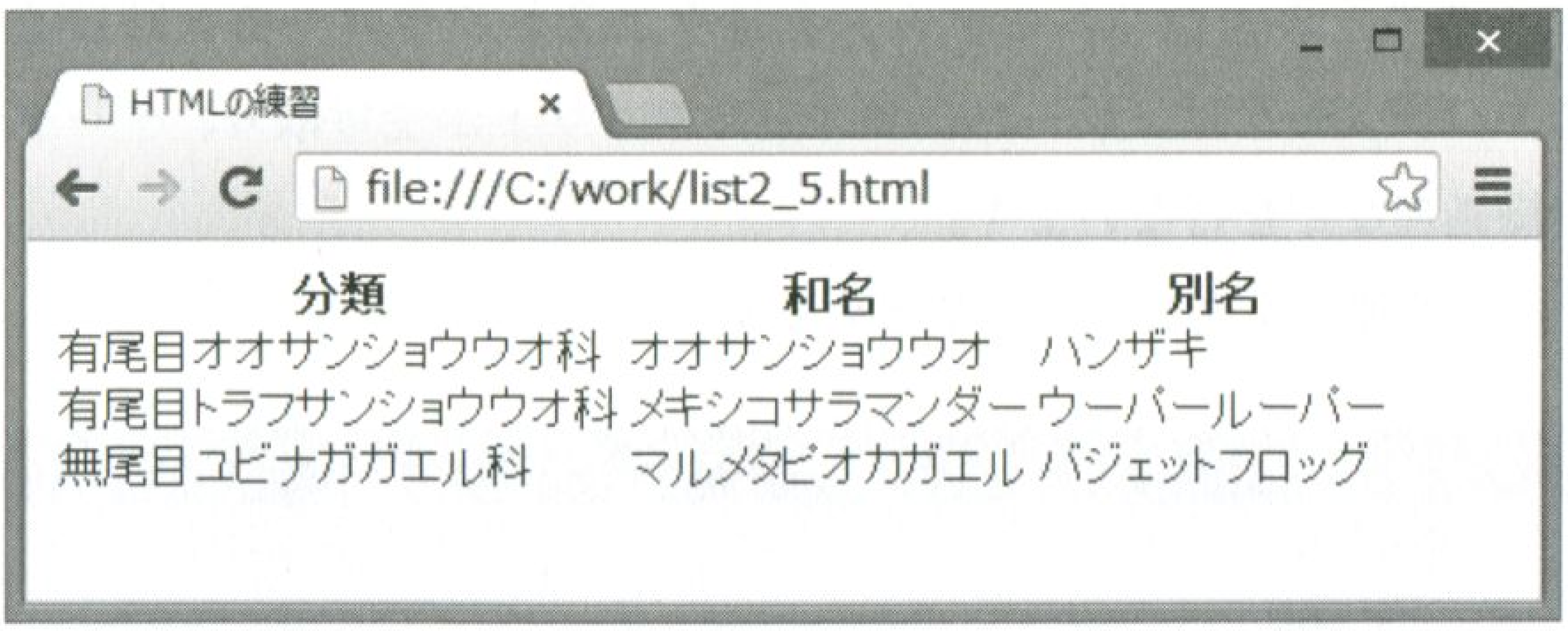
リスト2.5では、3列、4行（うち見出し1行）のテーブルを表示しています。

リスト2.5　テーブルの表示

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>HTMLの練習</title>
 </head>
 <body>
  <table>
   <tr>
    <th>分類</th>
```

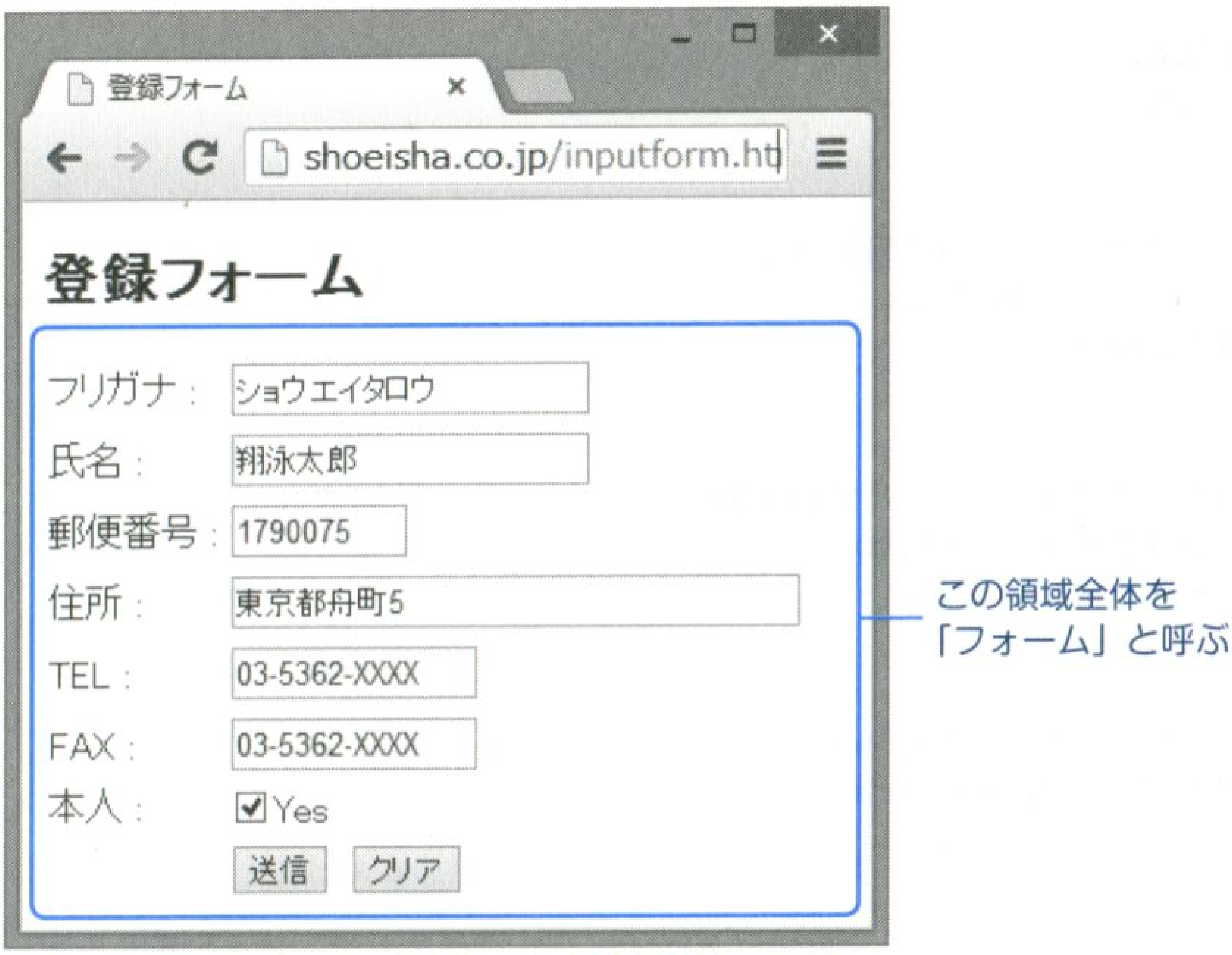
```
      <th>和名</th>
      <th>別名</th>
    </tr>
    <tr>
      <td>有尾目オオサンショウウオ科</td>
      <td>オオサンショウウオ</td>
      <td>ハンザキ</td>
    </tr>
    <tr>
      <td>有尾目トラフサンショウウオ科</td>
      <td>メキシコサラマンダー</td>
      <td>ウーパールーパー</td>
    </tr>
    <tr>
      <td>無尾目ユビナガガエル科</td>
      <td>マルメタピオカガエル</td>
      <td>バジェットフロッグ</td>
    </tr>
  </table>
 </body>
</html>
```

■実行結果



フォーム

フォームとは、文字を入力するテキストボックスや送信ボタンなど、ユーザーに情報を入力させるための部品をまとめたもののことです。ショッピングサイトでよく見られる、ユーザーの個人情報を入力するページなどはフォームで作られています。



フォームはさまざまな入力部品をまとめるための領域

フォーム内の送信ボタンがクリックされると、入力されたデータがサーバーに送られます。このとき送信されるデータを**リクエストパラメータ**と呼びます。リクエストパラメータは、サーバーサイドスクリプトが受け取り、データベースへの登録処理などを行ないます。

まず、フォーム関連のタグを以下にまとめます。

<table>
<tr><th>タグ名</th><th colspan="3">説明</th></tr>
<tr><td rowspan="7">form</td><td colspan="3">フォーム全体を囲む
▼formタグが持つ属性</td></tr>
<tr><th>属性名</th><th colspan="2">指定内容</th></tr>
<tr><td>name</td><td colspan="2">任意の名前</td></tr>
<tr><td>action</td><td colspan="2">送信先の URL を指定する</td></tr>
<tr><td rowspan="2">method</td><td colspan="2">フォームの入力データ（パラメータ）をどのように送信するかを指定する。指定値は以下</td></tr>
<tr><td>get
post</td><td>URL の後ろにパラメータを付加する
送信先URL?パラメータ名1=値&パラメータ名2=値…
パラメータを隠して送信する</td></tr>
<tr><td colspan="3">例 <form name="edit" action="entry.php" method="post">
　…
</form></td></tr>
</table>

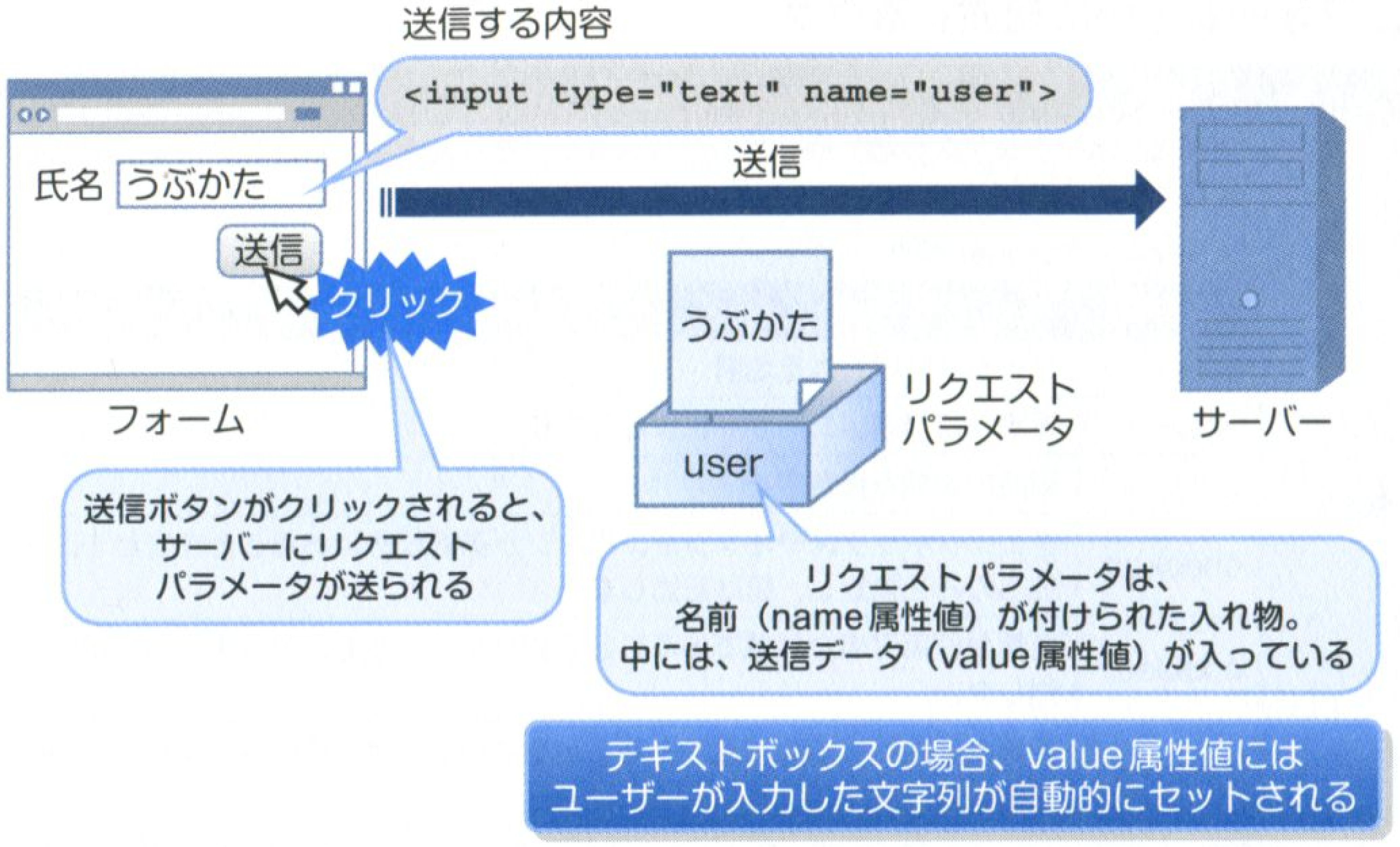
◆ フォームの中に記述するタグ

<table>
<tr><th>タグ名</th><th colspan="2">説明</th></tr>
<tr><td rowspan="9">input</td><td colspan="2">テキストボックス（入力欄）やオプションボタン（ラジオボタン）などの、入力部品を定義する
▼<input>タグが持つ属性</td></tr>
<tr><th>属性名</th><th>指定内容</th></tr>
<tr><td>name</td><td>サーバーに送られる名前</td></tr>
<tr><td>value</td><td>要素が持つ値。ユーザーによる入力値</td></tr>
<tr><td>type</td><td>部品の種類を指定する</td></tr>
<tr><td>checked</td><td>チェックボックスやオプションボタンが選択状態にあることを表わす。属性名のみを指定し、値は記述しない</td></tr>
<tr><td>enabled</td><td>要素が使用可能な状態であることを表わす。属性名のみを指定し、値は記述しない</td></tr>
<tr><td>disabled</td><td>要素が使用不可能な状態であることを表わす。属性名のみを指定し、値は記述しない</td></tr>
<tr><td colspan="2">例 <input type="radio" name="man" value="1" checked></td></tr>
<tr><td rowspan="2">label</td><td colspan="2">入力部品の項目名を定義する</td></tr>
<tr><td colspan="2">例 <label>氏名<input type="text" name="username"></label></td></tr>
</table>

◆ <input>要素のtype属性に指定できる値の一覧

値	種類	説明	表示
type="text"	テキストボックス	文字を入力するための部品	
type="checkbox"	チェックボックス	複数選択可能な部品	☐ ☑
type="radio"	オプションボタン（ラジオボタン）	1件のみ選択可能な部品	○ ◉
type="hidden"	隠しフィールド	Webページ上には表示されない部品。スクリプトで使うデータを保持しておきたい場合などに使う	表示されない
type="submit"	送信ボタン	クリックするとフォーム内の入力データが送信される	送信
type="reset"	リセットボタン	クリックするとフォーム内の入力データがすべてクリアされる	リセット
type="button"	汎用ボタン	クリックしても何も起こらないボタン。スクリプトを設定して使う	ボタン

次の図は、フォームの一般的な利用イメージです。ユーザーがフォームで入力した情報は、リクエストパラメータとしてサーバーへ送信されます。



フォームの利用イメージ

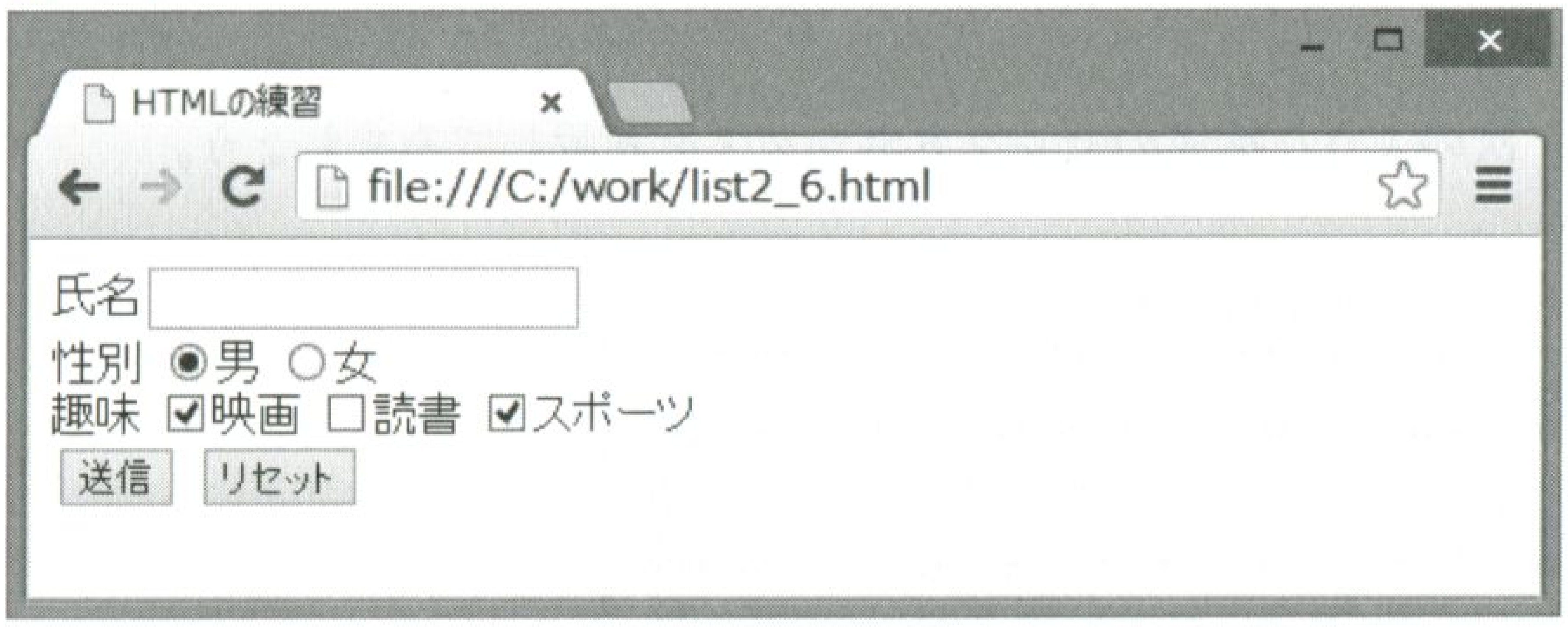
リスト2.6は、単純な入力フォームを定義した例です。

リスト2.6　フォームの表示

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>HTMLの練習</title>
 </head>
 <body>
  <form>
   <label>氏名<input type="text" name="username"></label>
   <br>
   <label>性別
    <input type="radio" name="gender" value="1" checked>男
    <input type="radio" name="gender" value="2">女
   </label>
   <br>
   <label>趣味
    <input type="checkbox" name="hobby" value="1" ⏎
checked>映画
    <input type="checkbox" name="hobby" value="2">読書
    <input type="checkbox" name="hobby" value="3" ⏎
checked>スポーツ
   </label>
```

```
    <br>
    <input type="submit" value="送信">
    <input type="reset" value="リセット">
   </form>
  </body>
 </html>
```

■実行結果



注意

Webページ内の改行について

HTMLで記述した文章をブラウザで表示すると、改行やインデントなどは無視されて表示されます。改行をページに表示したい場合は、該当箇所に
要素を記述します。ただし、本来
要素は、リスト2.6のようにレイアウトを整える用途では利用しません。
は長い文章を折り返す目的で、主に<p>要素の中などで利用します。見た目を整える場合は、
要素を挿入するのではなく、次章で取り上げるCSSを利用します。

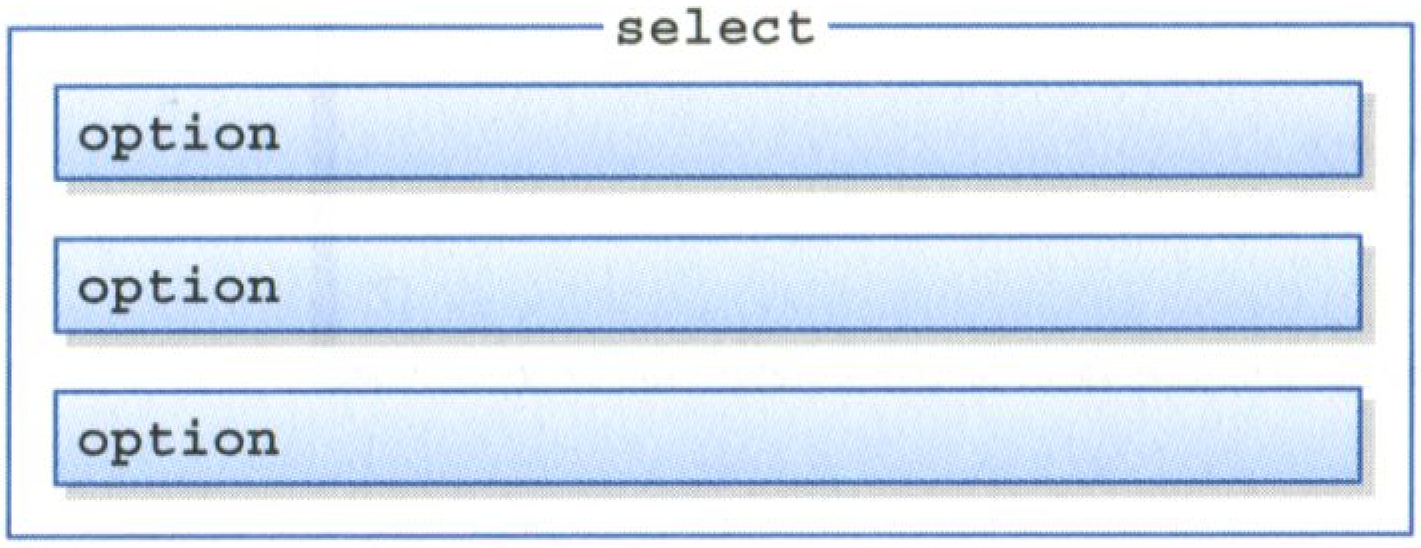
この他にも、フォーム内に記述する要素として、複数タグの組み合わせによるセレクトボックスがあります。これについては次項で紹介します。

セレクトボックス

セレクトボックスは、複数の選択肢の中から、ユーザーによる選択を受け付けます。<input>要素同様、フォーム内の入力部品の一種です。

<table>
<tr><th>タグ名</th><th colspan="2">説明</th></tr>
<tr><td>select</td><td colspan="2">セレクトボックス全体を囲む</td></tr>
<tr><td rowspan="5">option</td><td colspan="2">選択肢を定義する</td></tr>
<tr><td colspan="2">▼<option>タグが持つ属性</td></tr>
<tr><th>属性名</th><th>指定内容</th></tr>
<tr><td>value</td><td>入力データとして送信される値</td></tr>
<tr><td>selected</td><td>項目が選択中であることを表わす。属性名のみを指定する</td></tr>
</table>

例として、次のような構成のセレクトボックスを表示してみましょう。



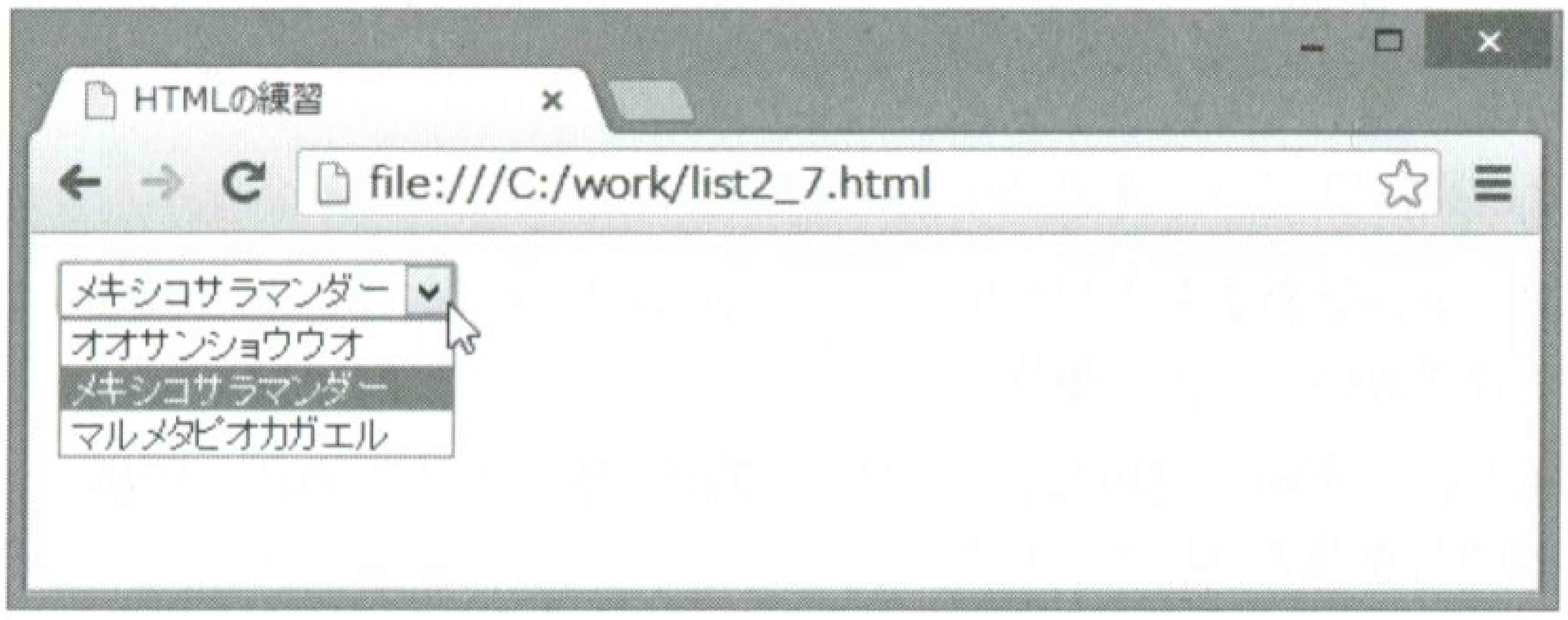
セレクトボックスの構成イメージ

リスト2.7では、3件の項目を持つセレクトボックスを表示しています。

リスト2.7　セレクトボックスの表示

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>HTMLの練習</title>
 </head>
 <body>
  <form>
   <select>
    <option value="1">オオサンショウウオ</option>
    <option value="2" selected>メキシコサラマンダー</option>
    <option value="3">マルメタピオカガエル</option>
   </select>
  </form>
 </body>
</html>
```

■実行結果



Point!

▶ 複数のタグを組み合わせることにより、1つの部品（リスト、セレクトボックス、テーブル、フォームなど）を生成するタグがある。



この章の到達度チェック

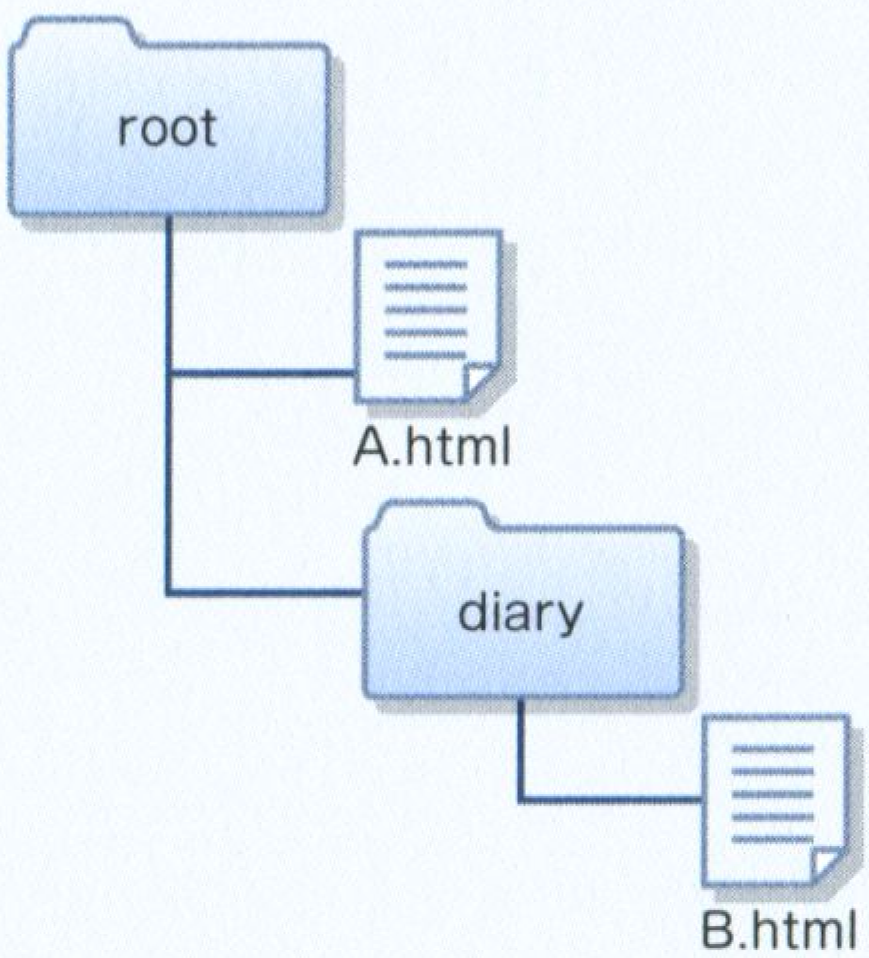
Q1 HTML5で推奨されている文字コードはなんですか？

Q2 オプションボタンを表示するためには、<input>タグにどのような属性と値を付ければ良いでしょうか？

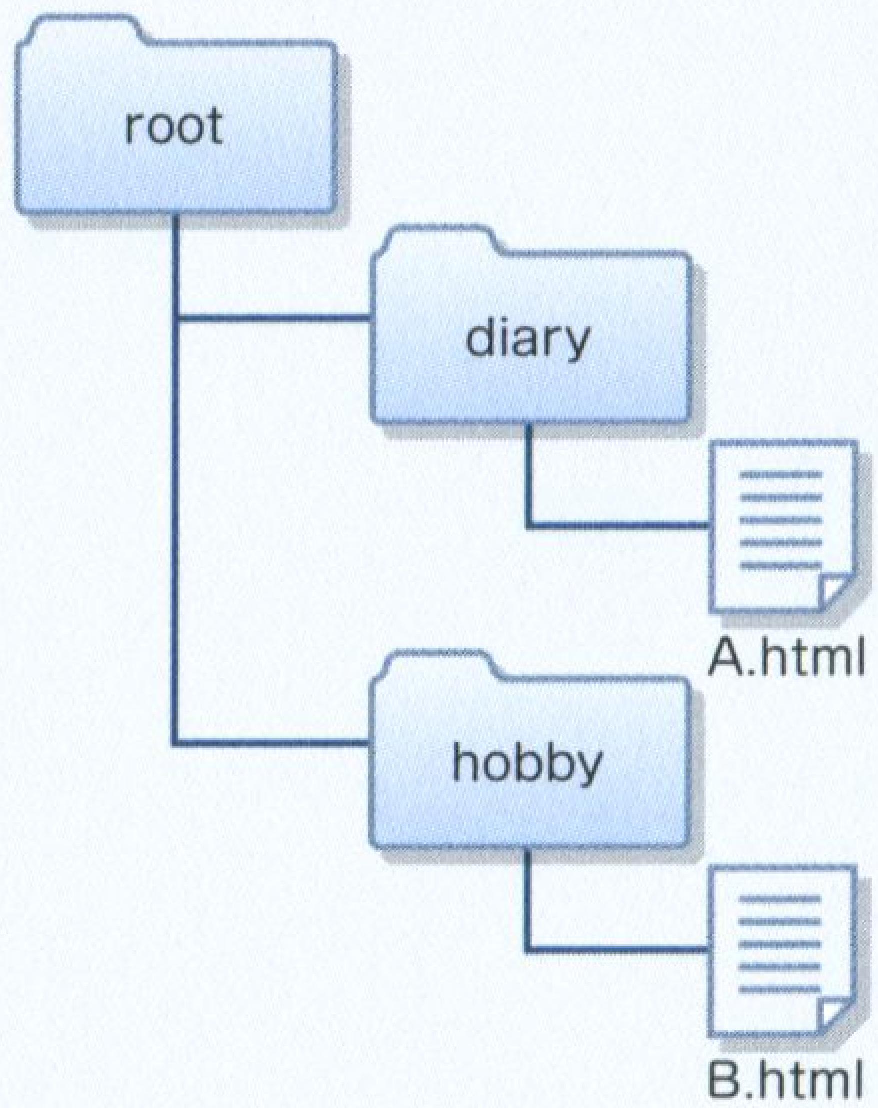
Q3 縦2列、横3行（先頭が見出し行）のテーブルを生成するために、空欄A～Cを適切なタグで埋めてください。

```
<table>
<[ A ]><[ B ]>No.<[ B ]><[ B ]>商品名</[ B ]></[ A ]>
<[ A ]><[ C ]>001</[ C ]><[ C ]>えんぴつ</[ C ]></[ A ]>
<[ A ]><[ C ]>002</[ C ]><[ C ]>消しゴム</[ C ]></[ A ]>
</table>
```

Q4 A.htmlからB.htmlへ相対パス指定で移動する場合、リンク先URLはどのように記述しますか？



Q5 A.htmlからB.htmlへ相対パス指定で移動する場合、リンク先URLはどのように記述しますか？



➲解答は巻末に掲載

第3章

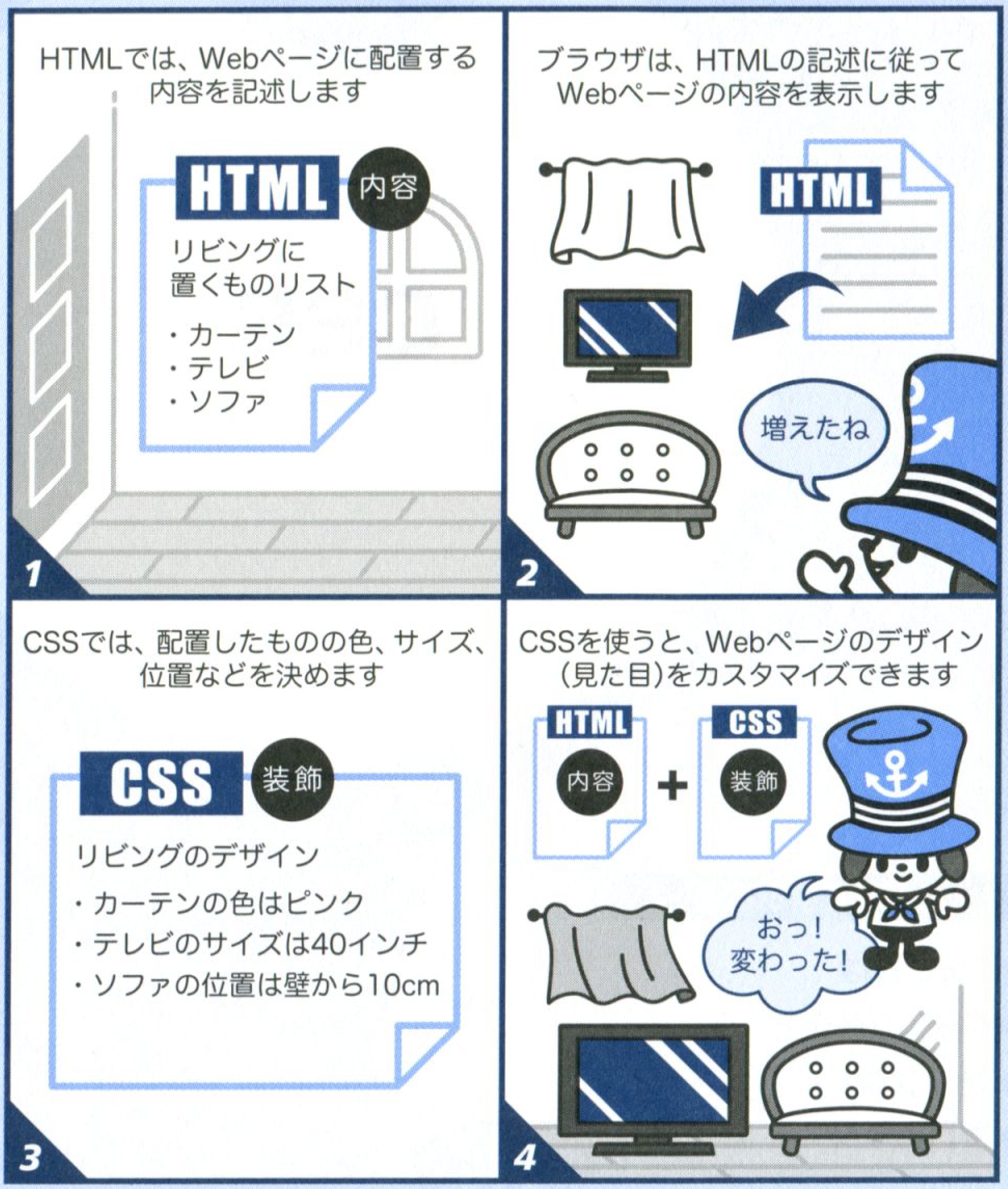
CSS

この章で学ぶこと

- CSSとは
- CSSの書き方
- セレクタ
- プロパティ



● HTMLで内容を決め、CSSでデザインする ●

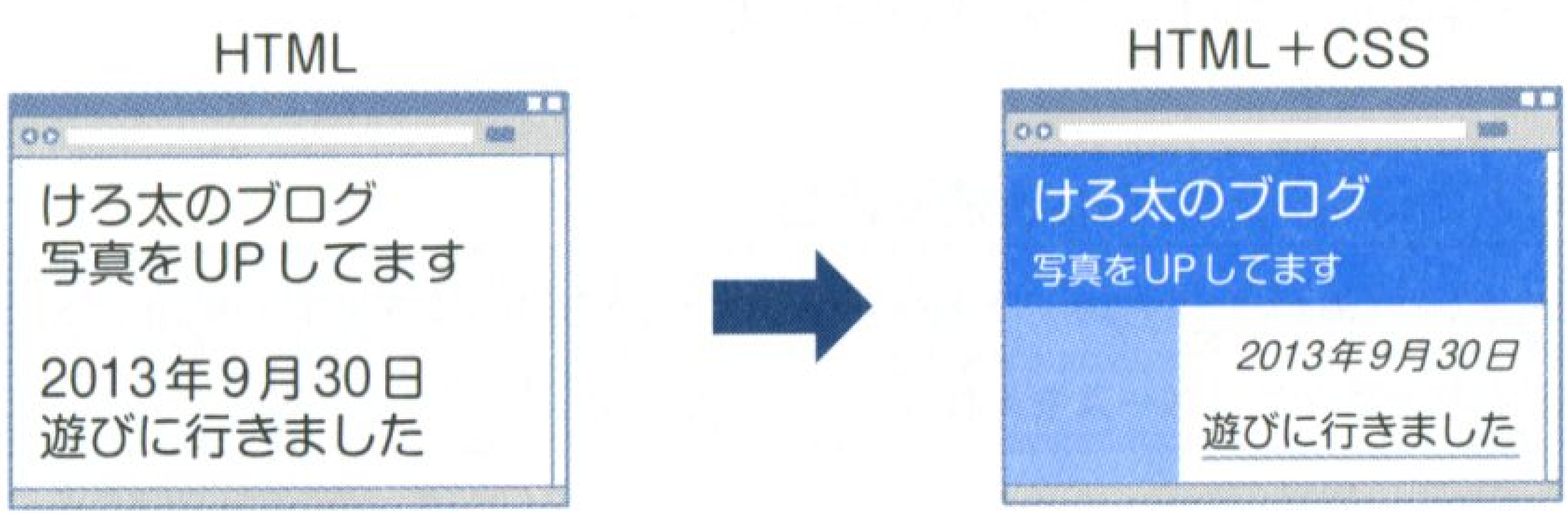


第2章で学んだHTMLは、文書を構造的に管理するための技術でした。本章で学ぶCSSをHTMLと組み合わせて使うことで、文書を装飾できます。CSSの操作は、JavaScriptによる処理の定番です。

3-1 CSSの基礎知識

CSSとは

CSS（シーエスエス）（Cascading Style Sheets）は、HTML文書を装飾する（飾り付ける）ための言語です。背景や文字の色設定を行なったり、文字や画像のサイズや表示位置を調整したりと、Webページに**スタイル**（見栄えに関する情報）を適用するために使われます。




```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<title>けろ太のブログ</title>
</head>
<body>
<div>
<h1>けろ太のブログ</h1>
<p>写真をUPしてます</p>
</div>
<div>
<h2>2013年9月30日</h2>
<p>遊びにいきました。</p>
</div>
</body>
</html>
```

```
*{
 margin: 0;
 padding: 0;
}
body{
 background-color: #66ccff;
}
h1{
 font-size: 14px;
}
h2{
 font-size: 10px;
 font-style: italic;
}
div{
       ⋮
```

HTML＋CSSで装飾された静的Webページを作成できる

Note HTML文書の見た目を決める仕組みを**スタイルシート**と呼びます。CSSは、**スタイルシート言語**の1つです。スタイルシート言語は、HTMLなどのマークアップ言語で記述された文書のスタイルを指定するために使われます。

3

CSSを記述する場所

HTMLの中にCSSを記述する方法もありますが、基本的にHTMLとCSSは別々のファイルに記述します。Webページの本体はHTMLファイルなので、HTMLファイルから、CSSで記述したファイル（CSSファイル）を読み込みます。

1. CSSのコードのみをファイルに記述し、拡張子を「.css」として保存します。
2. HTMLファイルに<link>要素を記述し、href属性にCSSファイルのURLを指定します。<link>要素は<head>要素の中に記述します。なお、<link>要素は、終了タグのない空要素です。

文法 <link>タグでCSSファイルを読み込む

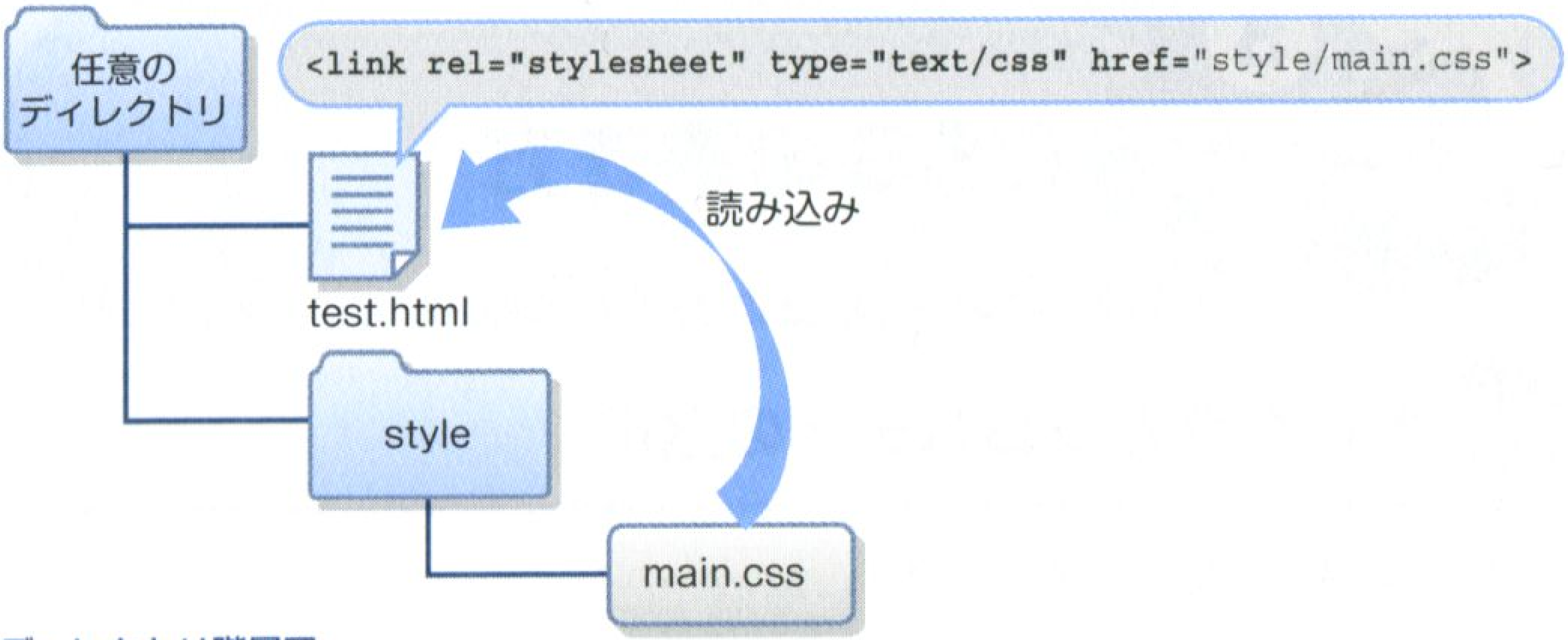
```
<link rel="stylesheet" type="text/css" href="CSSファイルのURL">
```

属性の前には半角スペースを入れずに！

rel属性とtype属性については、CSSファイルを読み込む場合は必ずこのように書く、決まりごとだとおぼえてください。CSSファイルのURLは、絶対パスまたは相対パスで指定します。

一般的にHTMLファイルにCSSファイルを読み込む場合、右ページの図のようにファイルを配置し、次のように記述します。

```
<link rel="stylesheet" type="text/css" href="style/main.css">
```



ディレクトリ階層図



Point!

- CSSはHTML文書を装飾するために用いる。
- CSSコードは拡張子「.css」のファイルとして保存する。
- HTMLファイルからCSSファイルを読み込む。

■ Check Test 1

Q1 CSSとはなにか説明してください。

Q2 次はHTMファイルからCSSファイルを読み込むタグです。空欄A～Dを適切な語で埋めてください。

```
< [ A ] rel=[ B ] type=[ C ] [ D ]="CSSファイルのURL">
```

➲解答は巻末に掲載

3-2 CSSの書き方

セレクタとプロパティの指定

CSSを記述する際に必要な情報は、

- どの要素（タグ）に対して
- どのようなスタイルを適用するのか

の2点です。

「どの要素（タグ）に対して」を**セレクタ**で指定し、「どのようなスタイルを適用するのか」を**プロパティ**で指定します。プロパティは色やサイズなど、さまざまな種類があるため、1つのセレクタに対して複数のプロパティを付与できます。

文法 セレクタとプロパティの記述

```
セレクタ { プロパティ: 値 }
```

プロパティ（スタイルの種類）には必ず値を指定する

プロパティと値はコロン（:）で区切る

```
セレクタ {
    プロパティ: 値;
    プロパティ: 値;
        ⋮
}
```

最後のプロパティ指定の;は省略できる。ただし、複数指定する場合は必ず;で終わる、とおぼえたほうが間違いがない

複数のプロパティを指定する場合は、セミコロン（;）で区切って改行する

■記述例

```
p {
  color: red;
  font-size: 10px;
}
```

<p>要素に対して

colorプロパティの値をred（赤）に指定

font-sizeプロパティの値を10px（10ピクセル）に指定

注意 HTML同様、CSSを記述するときには、以下の注意点があります。
- 必ず半角英数字で記述する
- 英字の大文字と小文字の区別はないが、小文字で統一したほうが良い

プロパティには必ず値を指定し、プロパティと値は必ずコロン（:）で区切ります。プロパティや値は、英単語の意味がわかれば、ある程度類推できるでしょう。colorは「色」、font-sizeは「文字サイズ」を指定するプロパティです。記述例では、<p>要素、つまり段落全体に対して、文字の色を赤、文字のサイズを10ピクセルというスタイルを適用しています。

次節では、よく使われるセレクタとプロパティについてまとめます。

Point!

- ▶ 1つのセレクタに対して複数のプロパティを設定することができる。
- ▶ スタイルの適用対象をセレクタで指定し、適用するスタイルをプロパティで指定する。
- ▶ プロパティと値は必ずコロン（:）で区切り、複数のプロパティを指定する場合はセミコロン（;）で区切る。



■ Check Test 2

Q1 次はCSSを記述するときの文法です。空欄AとBを適切な語で埋めてください。

```
[ A ] {
    [ B ] : 値;
    [ B ] : 値;
          ⋮
}
```

Q2 font-sizeはどのようなプロパティですか？

➲解答は巻末に掲載

3-3 セレクタとプロパティ

プロパティの種類

まずはプロパティからです。スタイルを適用するためのプロパティが数多く用意されています。

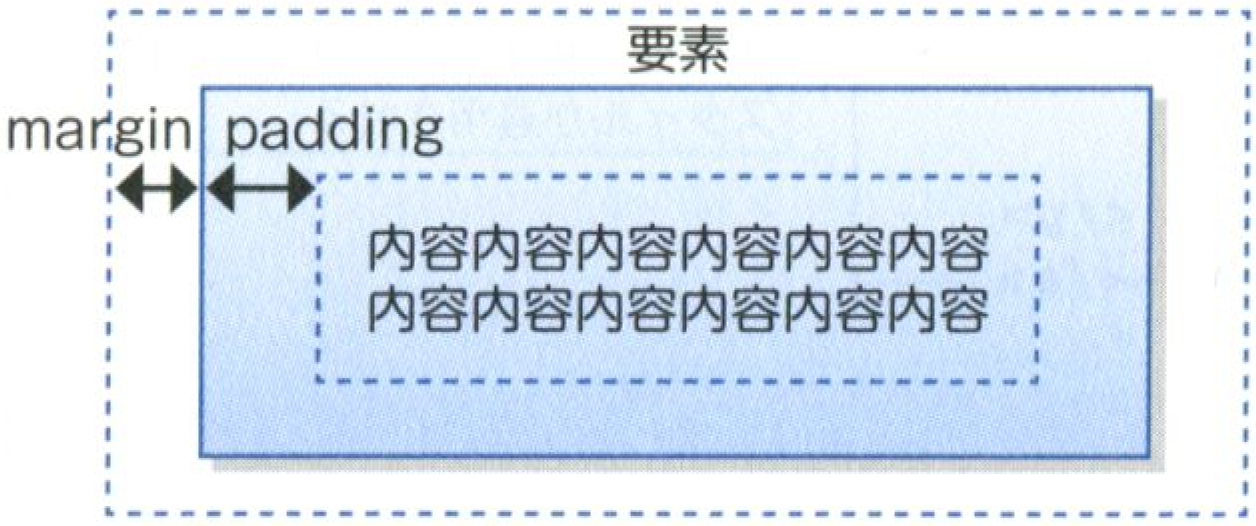
プロパティ名	説明	値
color	文字色を指定する	色名称（red、blue など）またはカラーコード（#000000 など）
background-color	背景色を指定する	
display	表示状態を指定する	block　要素を表示する none　要素を表示しない
font-size	文字サイズを指定する	次の単位による指定が可能 mm　ミリメートル cm　センチメートル in　インチ pt　ポイント pc　パイカ px　ピクセル em　英字「M」の高さを 1 とする単位 ex　英字「x」の高さを 1 とする単位 %　ウィンドウサイズまたは親要素のサイズを 100% とする単位
margin	要素の外側の余白を指定する	
padding	要素の内側の余白を指定する	
width	要素の幅を指定する	
height	要素の高さを指定する	
top	上端から要素までの距離を指定する	
left	左端から要素までの距離を指定する	

Note

カラーコードは、Photoshop、Illustrator、DreamweaverなどのWebデザイン制作に利用するグラフィックツールで見ることができます。コンピュータのディスプレイに表示される色は、光の三原色（赤、緑、青）を混ぜ合わせて作られています。それぞれの色の強さを最小0から最大255までの数値で表わし、16進数にして並べた6桁の数値がカラーコードです。

marginとpaddingはどちらも余白の幅を指定するプロパティですが、余白を取る位置が異なります。次のように、marginが要素の外側、paddingが要素の

内側です。



margin と padding の違い

セレクタの種類

セレクタは対象要素を指定する方法で、複数の種類があります。タグ名が同じ要素すべてにスタイルを適用したい場合と、1つの要素に対してのみスタイルを指定したい場合とでは、利用するセレクタが異なります。状況に応じて適切なセレクタを選択しましょう。

◇ 基本セレクタ

対象要素をタグ名で指定します。

リスト3.1では、HTML文書中の<a>要素すべてに対してフォントの大きさを50pxにするスタイルを適用しています。なお、CSS中のコメント（説明文）は /* から */ の間に記述します。

リスト3.1　基本セレクタの利用

CSS　style.css

```
a {
    font-size: 50px;  /* フォントの大きさを50pxにする */
}
```

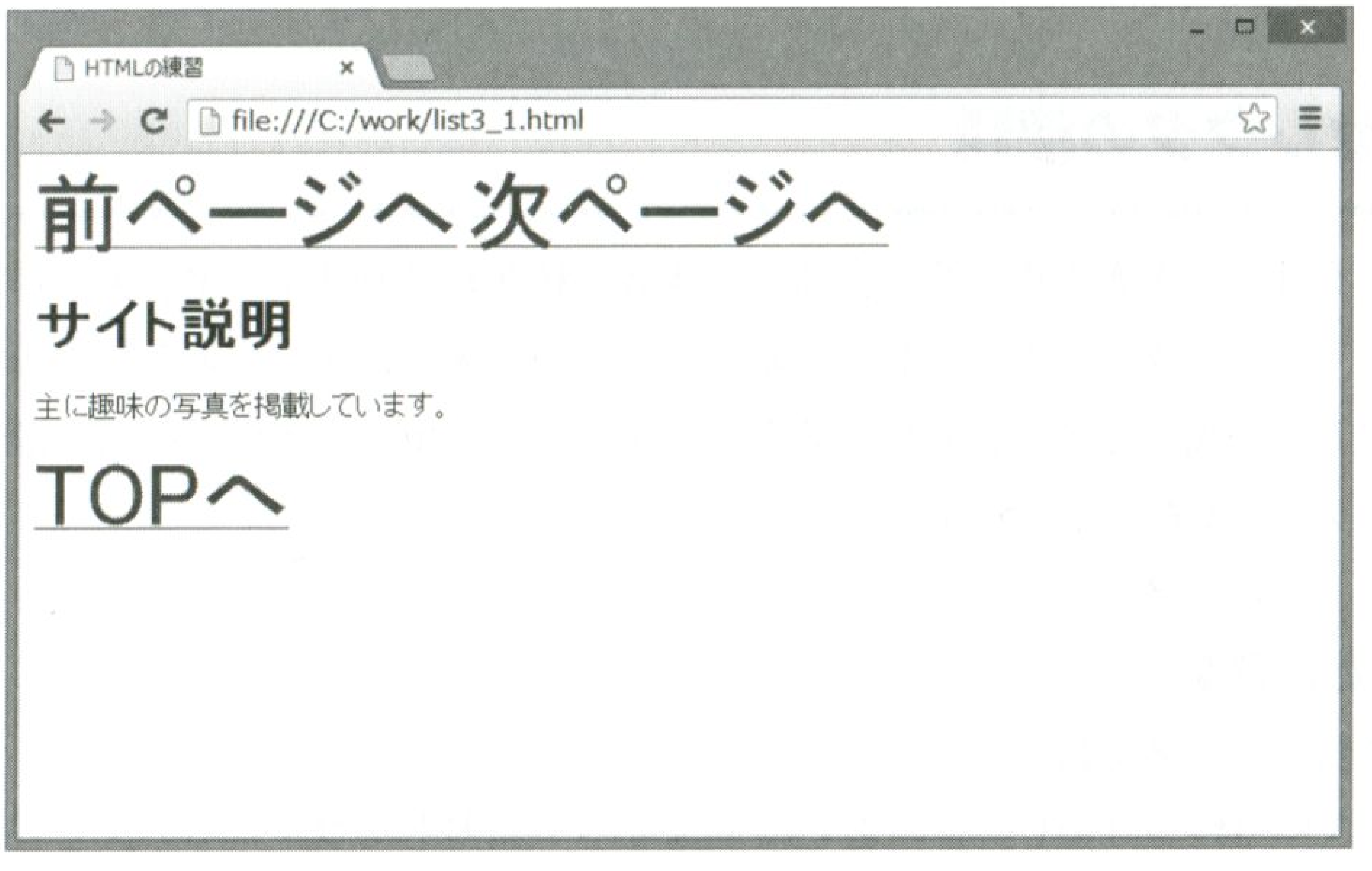
HTML　list3_1.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>HTMLの練習</title>
  <link rel="stylesheet" type="text/css" href="style.css">
```

```
</head>
<body>
 <a href="prev.html">前ページへ</a>
 <a href="next.html">次ページへ</a>
 <h1>サイト説明</h1>
 <p>主に趣味の写真を掲載しています。</p>
 <a href="index.html">TOPへ</a>
</body>
</html>
```

HTML文書中のaタグすべてにスタイルが適用される

■実行結果



◆ IDセレクタ

対象要素をID属性値で指定します。ID属性値はHTML文書の中で一意となる（重複する値を設定できない）ため、特定の要素1つだけにスタイルを適用したい場合に利用します。

文法 ID属性値の指定

```
#ID属性値 {
    プロパティ: 値;
}
```

ID属性値の前に#を付ける

リスト3.2では、ID属性の値として「next」を持つ要素に対し、スタイルを適用しています。

リスト3.2 IDセレクタの利用

CSS style.css

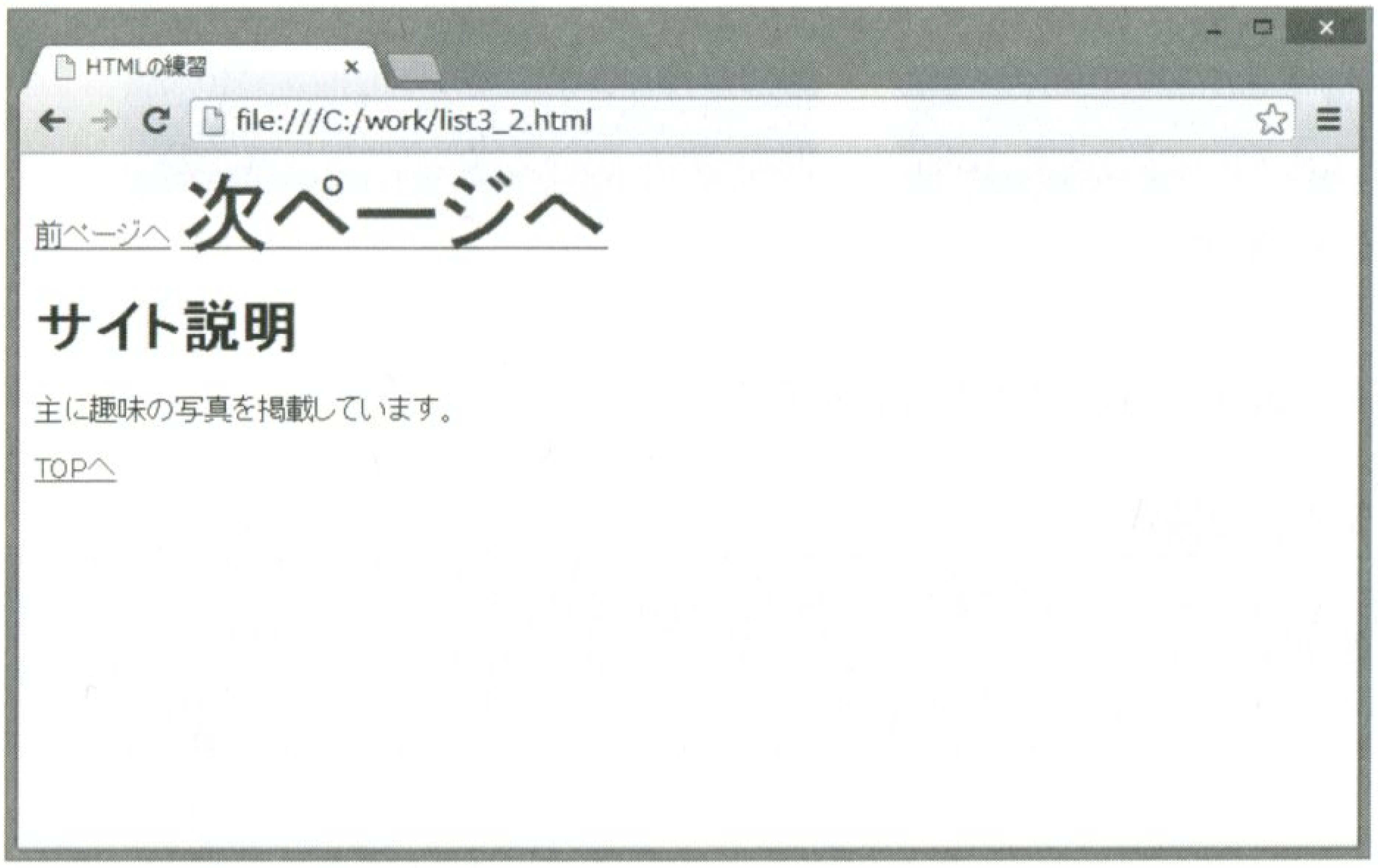
```
#next {
      font-size: 50px;
}
```

HTML list3_2.html（<body>要素内部のみ抜粋）

```
<a href="prev.html">前ページへ</a>
<a href="next.html" id="next">次ページへ</a>
<h1>サイト説明</h1>
<p>主に趣味の写真を掲載しています。</p>
<a href="index.html">TOPへ</a>
```

ID属性値に"next"が指定されている要素にのみスタイルが適用される

■実行結果



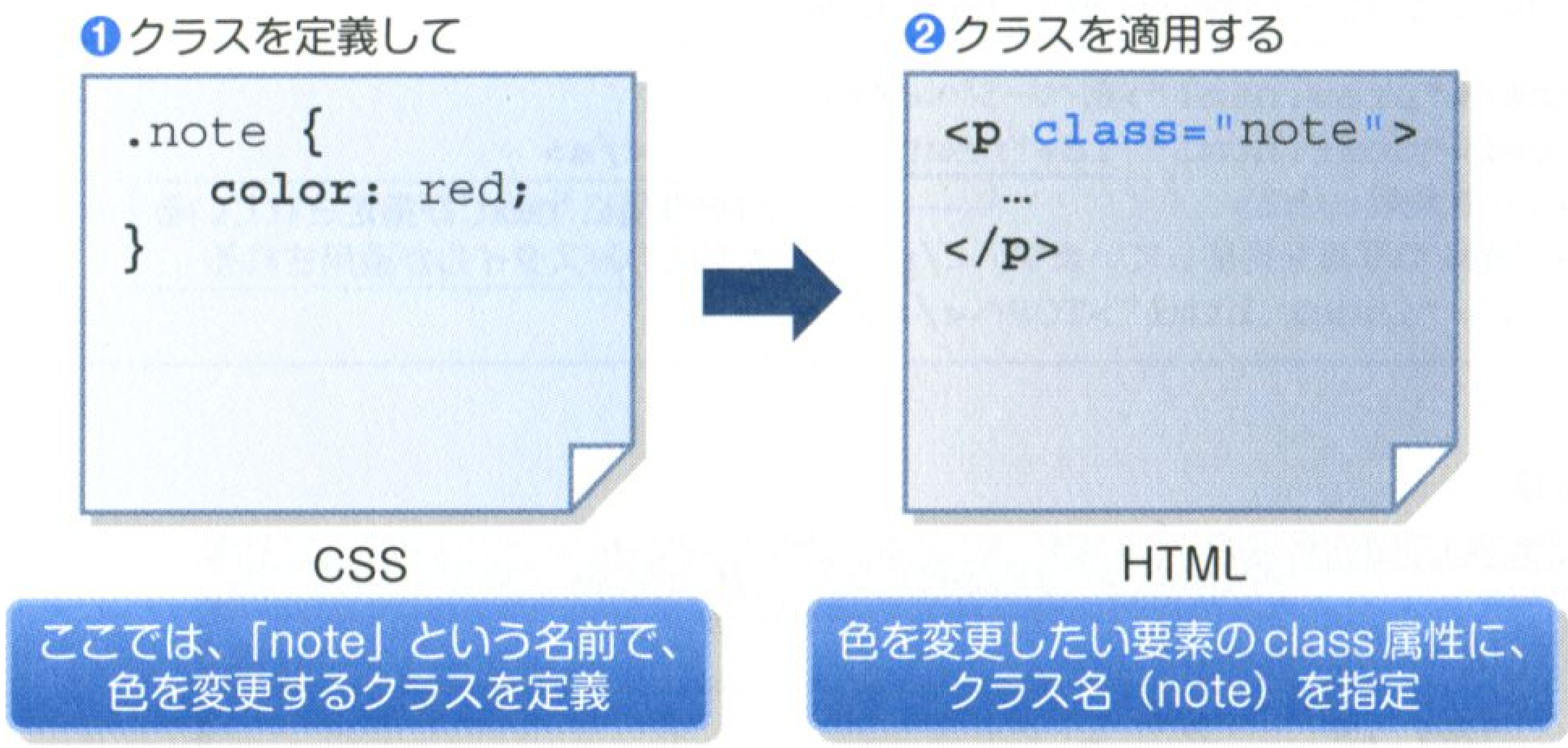
> **Note** HTMLの<body>要素以外の部分は、先ほどのリスト3.1と同じなので割愛しています。このコードを試す場合は、リスト3.1の<body>要素の内部をリスト3.2のHTMLに変更してください。以降の記述例でも、<body>要素以外の部分はリスト3.1と同じなので割愛します。

◇ クラスセレクタ

CSSにおける**クラス**とは、スタイルをひとまとめにして名前を付けたもののことです。クラスは、どの要素に対しても付けることができるので、複数の要

素の中から任意の要素を選択してスタイルを適用したい場合に利用します。

クラスセレクタの使い方は、その他のセレクタとは少し異なります。まずCSSでクラスを定義し、対象要素のclass属性値としてクラス名を指定します。



クラスセレクタの使い方

クラスを定義する方法は次のとおりです。

文法 クラス名の指定

```
.クラス名 {   ← クラス名の前にドット（.）を付ける
    プロパティ: 値;
}
```

ドット（.）に続けてクラス名を指定します。クラス名は任意の名称にすることができます。

リスト3.3では、CSSで「paging」という名前のクラスを定義し、「paging」クラスを適用したいHTMLの要素にclass属性で指定しています。

リスト3.3　クラスセレクタの利用

CSS　style.css

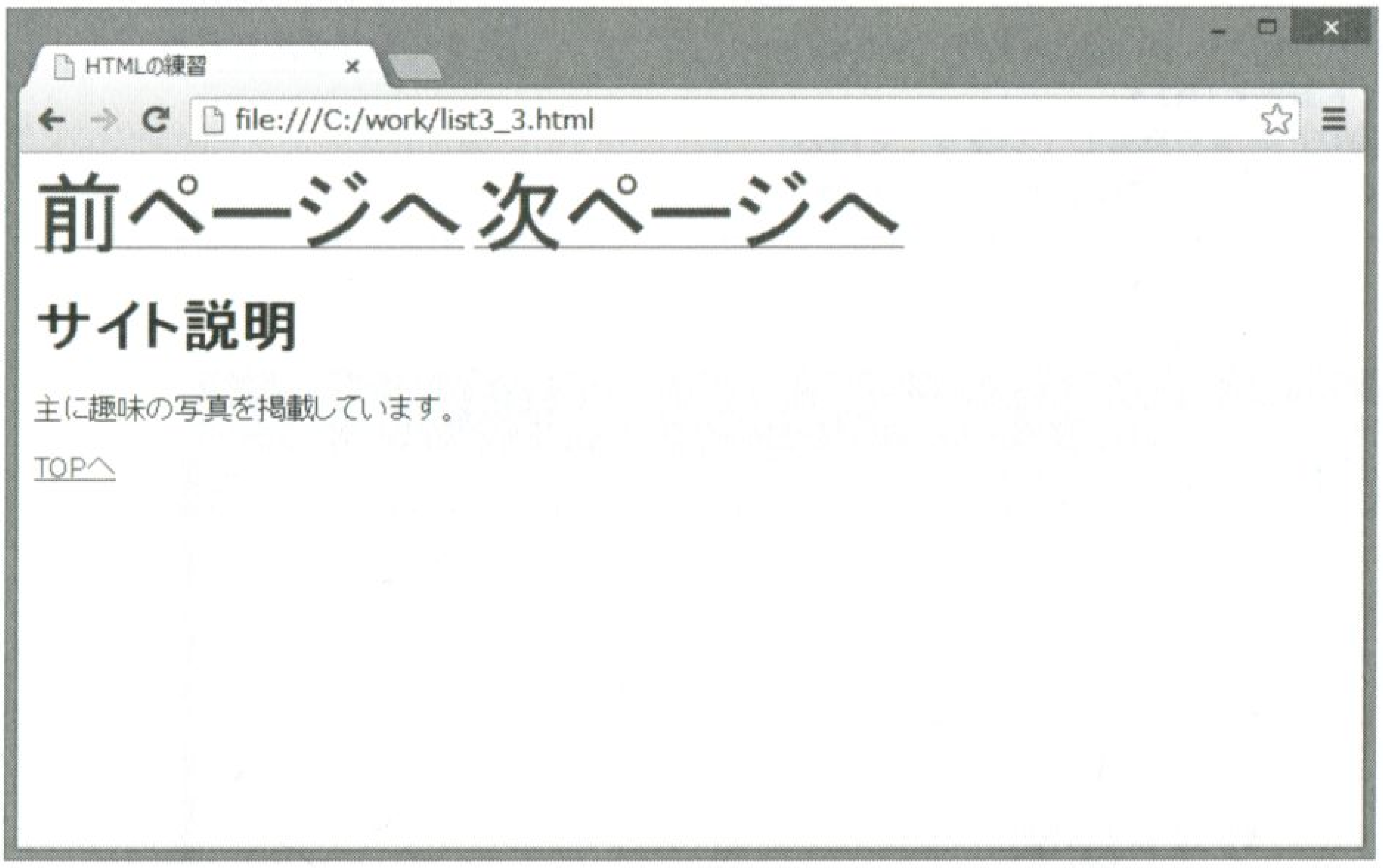
```
.paging {   ← クラス名はpaging
    font-size: 50px;
}
```

HTML list3_3.html（<body>要素内部のみ抜粋）

```
<a href="prev.html" class="paging">前ページへ</a>
<a href="next.html" class="paging">次ページへ</a>
<h1>サイト説明</h1>
<p>主に趣味の写真を掲載しています。</p>
<a href="index.html">TOPへ</a>
```

class属性値に"pading"が指定されている要素にのみスタイルが適用される

■実行結果



IDセレクタとクラスセレクタの違いはわかりにくいので、補足しておきます。基本的に、IDセレクタは「Webページの要素のうち1つだけにスタイルを適用したい場合」、クラスセレクタは「Webページの複数の要素に同時にスタイルを適用したい場合」に利用します。たとえば、スタイルを1つの画像にしか適用しない場合はIDセレクタを使い、複数件の画像にスタイルを適用したい場合はクラスセレクタを使います。

◇ 全称セレクタ

HTML文書中のすべての要素に対してスタイルを適用します。

リスト3.4では、HTML中の<body>要素内のすべての要素に対してスタイルを適用しています。

リスト3.4　全称セレクタの利用

CSS　style.css

```
* {        ← セレクタ名を＊にする
    margin: 50px;
}
```

HTML　list3_4.html（<body>要素内部のみ抜粋）

```
<a href="prev.html">前ページへ</a>
<a href="next.html">次ページへ</a>
<h1>サイト説明</h1>
<p>主に趣味の写真を掲載しています。</p>
<a href="index.html">TOPへ</a>
```

HTML文書中の要素すべてにスタイルが適用される

■実行結果



最後に使い分けをまとめておきましょう。

用途	利用するセレクタ	対象要素の指定方法	利用イメージ
要素にスタイルを適用	基本セレクタ	タグ名で指定	ページ内の同種の要素すべてに適用
特定の要素のみにスタイルを適用	ID セレクタ	タグに ID 属性値を指定	ページ内の 1 種のタグのみに適用
	クラスセレクタ	タグに class 属性を指定	ページ内の複数の要素の中から任意の要素を選択して適用
すべての要素に一括でスタイルを適用	全称セレクタ	セレクタ名を * にする	ページ（<body> 要素）内のすべての要素に適用

この章の到達度チェック

Q1 同一ディレクトリ内のCSSファイル「style.css」をHTMLファイルに読み込むには<link>タグをどのように記述すれば良いですか？

Q2 文書内のすべての<p>要素の文字色を青に設定するCSSを記述してください。なお、青色を表わす属性値は「blue」です。

Q3 「ex」というID属性値を持つ要素の文字サイズを15pxに設定するCSSを記述してください。

Q4 文書内のすべての要素の余白を内側／外側ともに0pxに設定するCSSを記述してください。

Q5 要素のサイズを幅200px、高さ80pxに設定する「box」という名前のクラスをCSSで定義してください。

➲解答は巻末に掲載

第4章

基本文法

この章で学ぶこと

- 基本的な書き方
- 変数とデータ型
- 演算子
- 配列
- 連想配列



4

● JavaScriptで扱うデータは変数や配列に入れる ●



第3章までは、JavaScriptを学ぶための下地作りでした。この章からは、いよいよJavaScriptについて学んでいきましょう。



4-1 書き方の基本

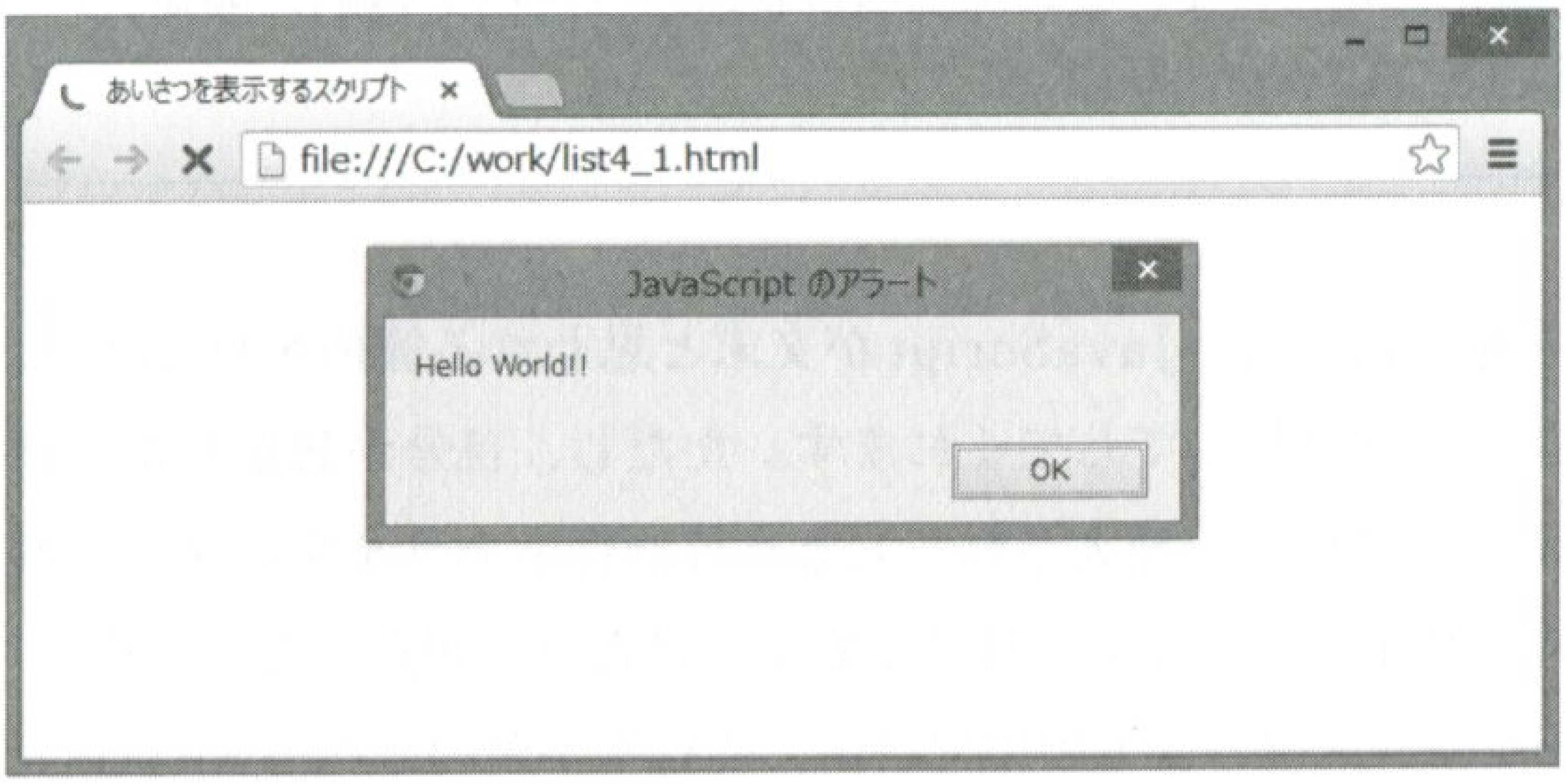
まずは1行のJavaScriptから！

書き方を学ぶ前に、簡単なJavaScriptを動かしてみましょう。リスト4.1をすべて半角の英数字と記号で入力してHTMLファイルとして保存し、ブラウザで表示してみてください。これまでに紹介したHTMLと異なるのは<script>から</script>までの3行です。そして、<script>タグに囲まれた❶の部分がJavaScriptです。

リスト4.1　ダイアログボックスの表示（list4.1.html）

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>あいさつを表示するスクリプト</title>
  <script>
   alert("Hello World!!");  ❶
  </script>
 </head>
 <body>
 </body>
</html>
```

■実行結果



ブラウザが起動すると、すぐにダイアログボックスが表示されるはずです。これはJavaScriptが表示したものです。[OK] ボタンをクリックすると、真っ白でなにも表示しないWebページだけが残ります（<body>要素の中になにも記述していないからです）。右上の × ボタンで閉じてください。

書き方のルール

<body>要素内に記述した内容を単純に表示するHTMLと違うことが、おわかりいただけたでしょうか。先ほどの例のJavaScriptのスクリプト部分を再掲します。

```
alert("Hello World!!");
```
この1行がJavaScript

まず、JavaScriptを記述する際のルールについて学びましょう。

JavaScriptの書き方のルール

ルール1 ▶基本的に半角の英数字と記号のみを使う。

ルール2 ▶文字列（文字の並び）を扱う場合はシングルクォート（'）かダブルクォート（"）で囲む。

ルール3 ▶大文字と小文字は別の文字として扱われる。

ルール4 ▶命令文の末尾にはセミコロン（;）を付ける。

ルール5 ▶複数行に渡るひとまとまりの命令群を波かっこ{ }で囲む。囲まれた範囲を**ブロック**と呼ぶ。

セミコロンを省略した場合、JavaScriptが文末と思われる箇所を探し出し、自動的にセミコロンを付け足してしてくれます。ただし、自分が想定した箇所とは異なる箇所にセミコロンが挿入されてしまう可能性もあります。文末を明確にするためにも、必ずセミコロンを付ける癖をつけたほうが良いでしょう。

また、JavaScriptは、スクリプトの中で自由に改行や半角のスペース（空白）、

タブを挿入して良いことになっています。たとえば、次の2つのスクリプトはどちらもまったく同じ動作をします。

```
for(i=0;i<10;i++){alert(i);}
```

```
for( i = 0 ; i < 10 ; i++ ) {
    alert( i );
}
```

{ から } までがブロック

どちらの書き方をするかは個人の好みにもよりますが、一般的には2つ目のようにブロックの中の命令文には先頭にタブを挿入して字下げ（**インデント**）をします。このようにすることで、どこからどこまでがブロックなのかがわかりやすくなります。

> Note このように、コードの改行やインデントなどが許されている（動作に関係しない）ことをフリーフォーマットと呼びます。JavaScriptやHTMLはフリーフォーマットの言語です。

JavaScriptを記述する場所

JavaScriptは、HTMLファイルの中に組み込む形で記述します。記述方法は、次の2通りがあります。

- <script>要素の中に記述する
- 外部のJavaScriptファイルに記述する（HTMLからJavaScriptを読み込む）

どちらの方法でも<script>要素を利用します。Webページをブラウザで表示すると、上から順にHTMLタグが読み込まれていきます。JavaScriptは<script>要素が読み込まれたタイミングで実行されます。

<script>要素は、<head>要素と<body>要素のどちらの中に記述してもかまいませんが、一般的には<head>要素の中に記述することが多いようです。

ただし、<script>要素の実行中は、以降のHTMLタグの読み込みがストップし、表示の遅延につながります。それを防ぐために<body>要素内の最下部に記述することもあります。

◆ <script>要素の中に記述する方法

<script>要素の開始タグと終了タグの間がJavaScriptを記述する位置です。

文法 JavaScriptの記述

```
<script>
    ここにJavaScriptを記述する
</script>
```

◆ 外部のJavaScriptファイルに記述する方法

<script>要素の中に直接JavaScriptを記述する方法では、HTMLによる文書構造の定義とスクリプトが混在してしまい、「ソースコードが読みづらい」「HTMLとスクリプトの各担当間の分業がしづらい」などのデメリットがあります。そこで、一般的にはHTMLファイルとJavaScriptファイルは別々に作成します。

まず、JavaScriptのスクリプトのみをファイルに記述し、拡張子を「.js」として保存します。次に、HTMLファイルに<script>タグを記述し、src属性にJavaScriptファイルのURLを指定します。このとき、開始タグと終了タグの間にはなにも記述しません。

文法 <script>タグでJavaScriptファイルを読み込む

```
<script src="JavaScriptファイルのURL"></script>
```

次は、リスト4.1のスクリプトを、外部のJavaScriptファイルに記述したときの例です。

リスト4.1A　JavaScriptを外部ファイルにした場合

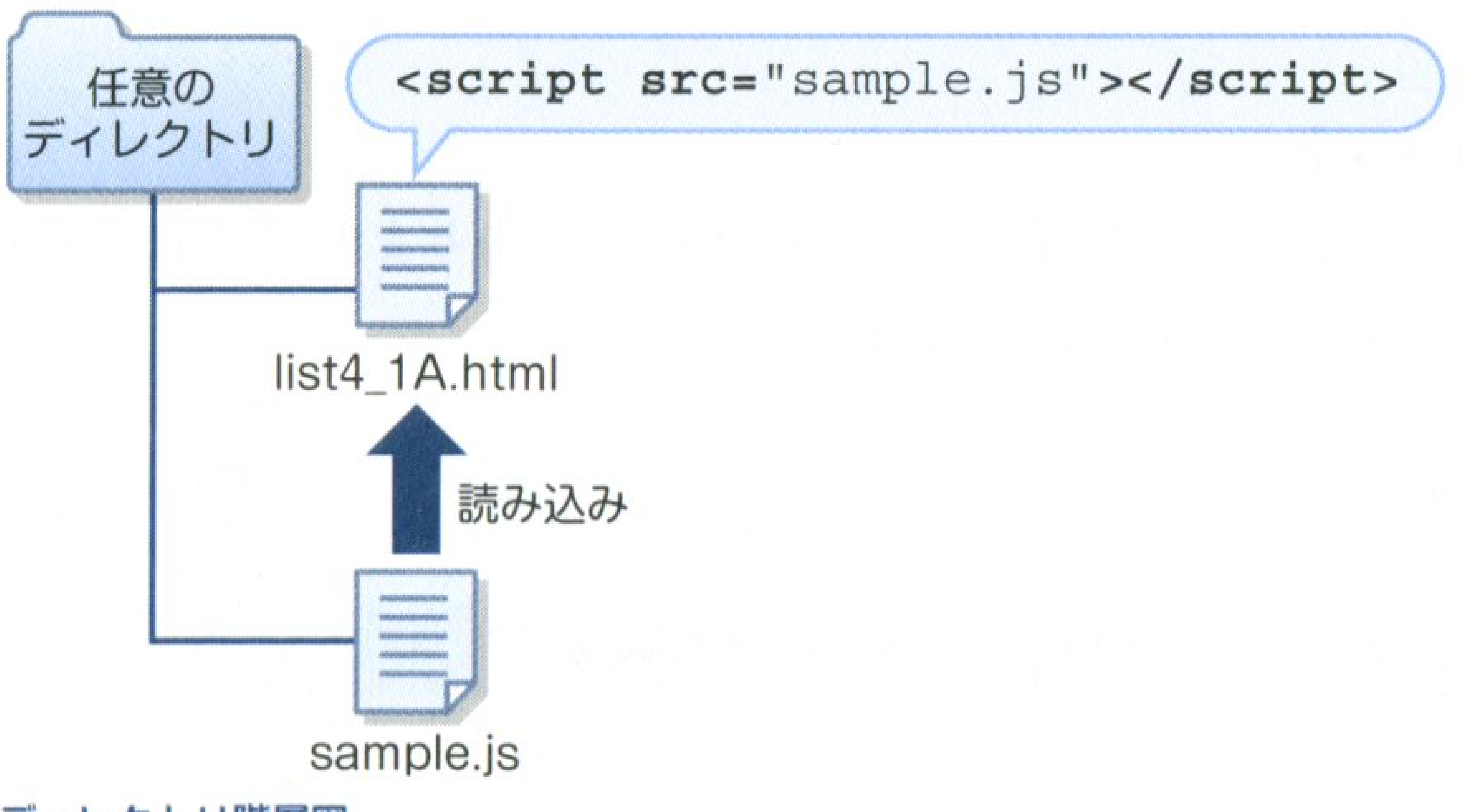
HTML　list4_1A.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>あいさつを表示するスクリプト</title>
  <script src="sample.js"></script>   ← JavaScriptファイルの読み込み
 </head>
 <body>
 </body>
</html>
```

JavaScript　sample.js

```
alert("Hello World!!");
```

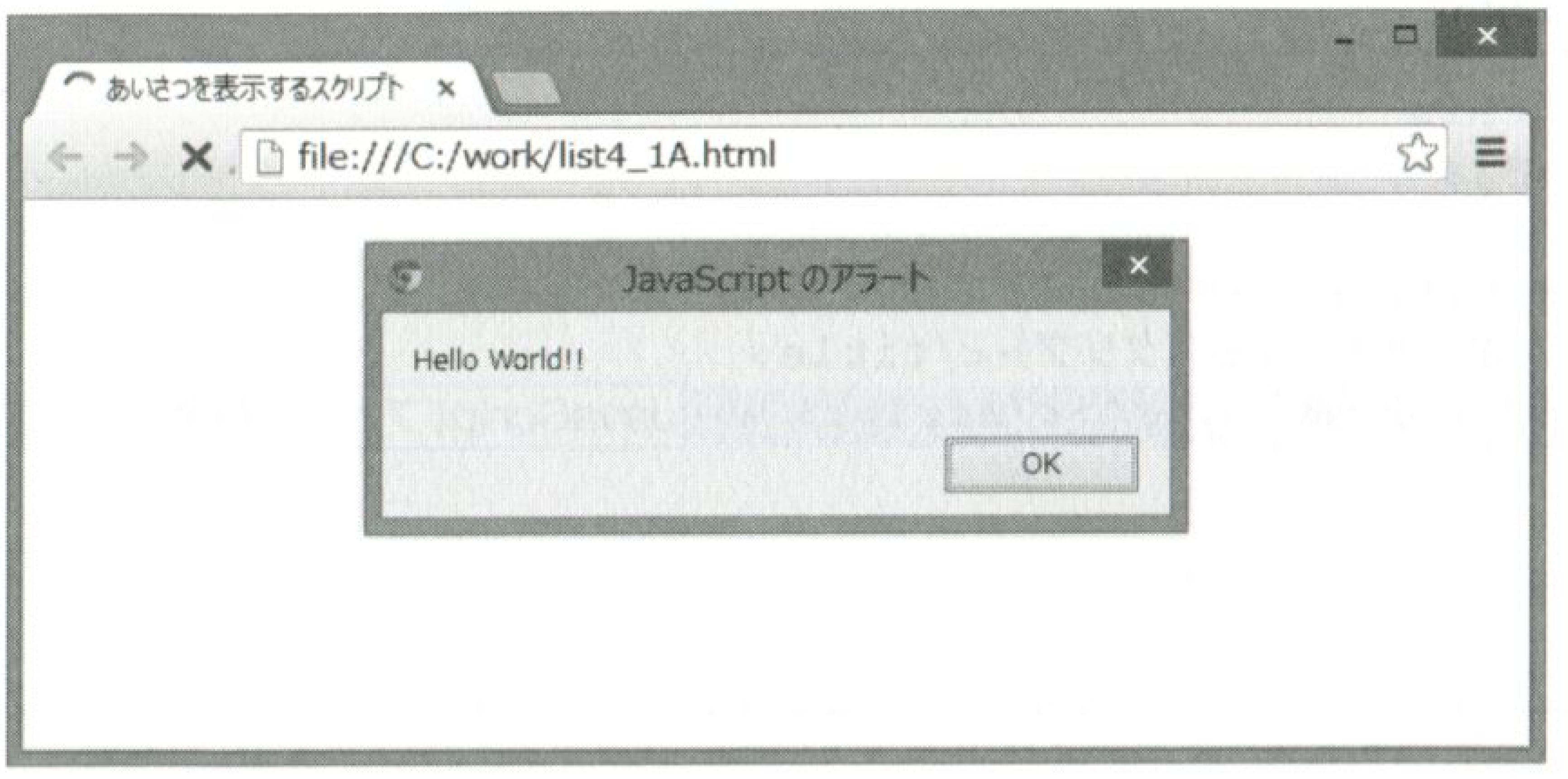
これは、HTMLファイルとJavaScriptファイルが、以下のように同一ディレクトリに配置されているときの例です。



ディレクトリ階層図

変更後のHTMLファイルをブラウザで表示すると、「list4_1A.html」に「sample.js」が読み込まれて、ダイアログボックスを表示するJavaScriptが実行されます。

■実行結果



コメント

コメントとは、スクリプトの中に自由に記述できるメモ書きのようなものです。コメントに使用できない文字はないため、全角文字を使って記述することもできます。

◇ 1行のコメント

//（スラッシュ2つ）よりも後ろの文字がコメントとなります。2つ目のように、命令文の後ろに記述することもできます。

```
//コメントです。
```

```
alert("Hello World!!");    //あいさつを表示する
```

◇ 複数行のコメント

/* と */ で囲んだ範囲がコメントになります。コメントは何行になってもかまいません。

```
/*
コメントとして記述した内容は、
スクリプトには影響しません。
*/
```

以上、2種類のコメントがあります。後でソースコードを見返したときや他の人が見たときに、何の処理を行なっているのかが一目見てわかるようにコメントを付けておくと良いでしょう。

値の出力

数字や文字列などの値(あたい)を表示して確認するには、alert(アラート)というJavaScriptの命令を使います。alertは、指定された値をダイアログボックスに表示する命令です。JavaScriptには、このように便利な命令群があらかじめ用意されています。

文法 ダイアログボックスの表示

```
alert("表示する値");
```

Note 72ページの「JavaScriptの書き方のルール」で紹介したように、JavaScriptで文字列を扱う場合はダブルクォート(")とシングルクォート(')のいずれかで文字列を囲みます。よって、上記の"表示する値"」の部分は、'表示する値'」とも記述できますが、本書ではダブルクォート(")を利用します。

◇ コンソールの利用

alertは値を確認する際に便利ですが、毎回ダイアログボックスが表示されるのもわずらわしいものです。そこで、以降ではブラウザの拡張機能であるコンソールを利用して値を表示します。

文法 コンソールへの値の表示

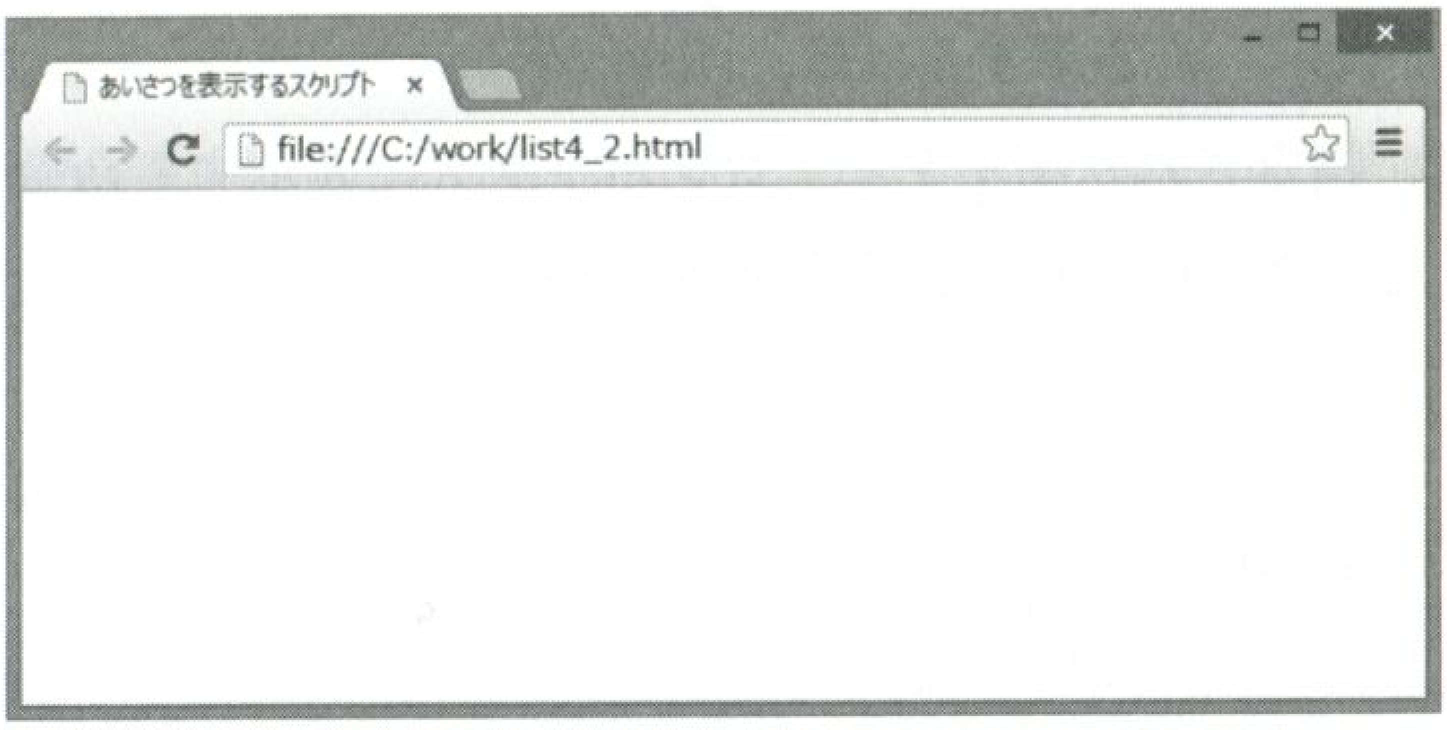
```
console.log("表示する値");
```

リスト4.1のJavaScriptを一部変更してみましょう。次のようにリスト4.1-❶の「alert」の部分を「console.log」に書き換えます。書き換えたHTMLファイルは「list4_2.html」という別名で保存してください。

リスト4.2 コンソールへの表示

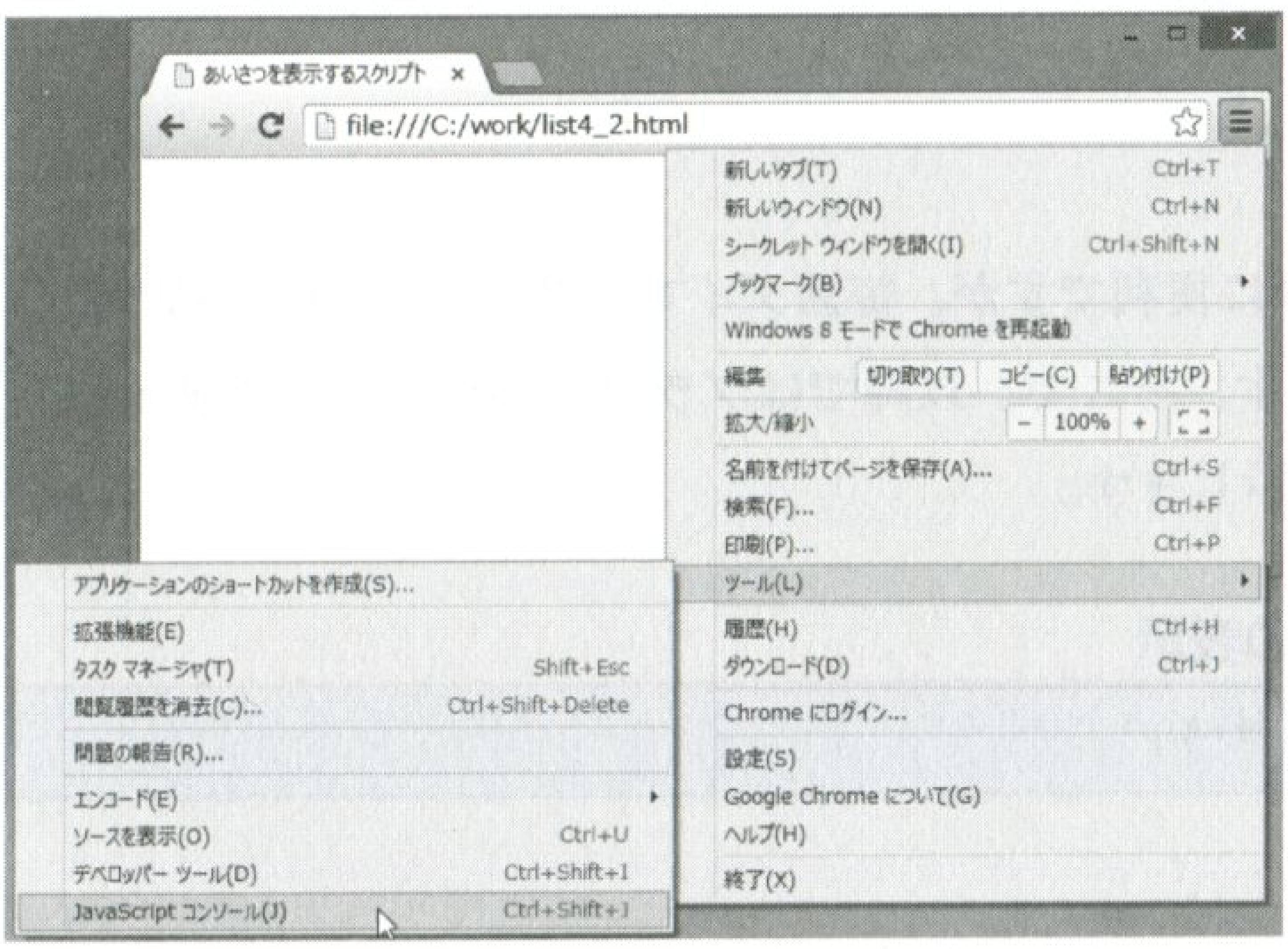
```
console.log("Hello World!!");    //あいさつをコンソールに表示
```

書き換えたらブラウザでHTMLファイルを表示してみましょう。次のような真っ白なWebページが表示されるはずです。



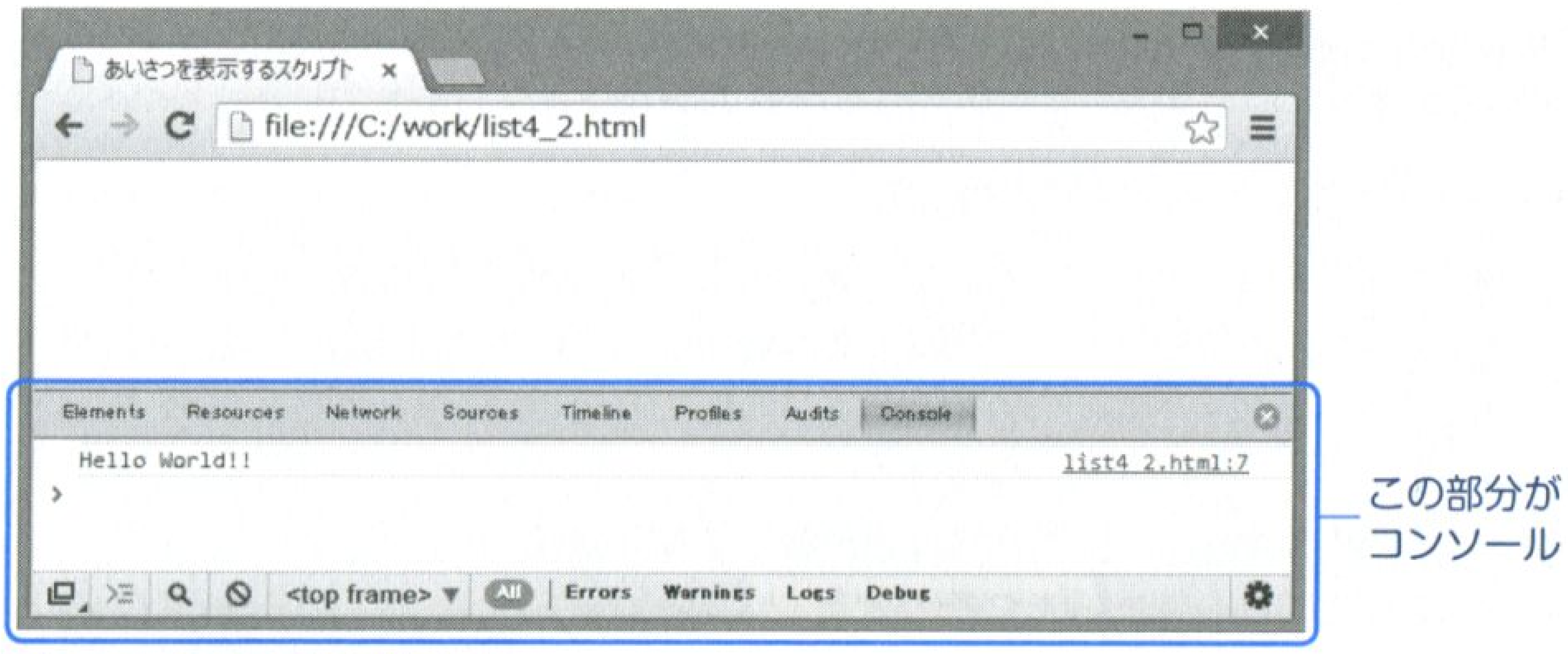
「console.log("Hello World!!");」に書き換えたWebページ

続いて、コンソールを表示します。Chromeの場合は、右上の三本線のアイコン≡をクリックし、表示されたメニューで［ツール］→［JavaScriptコンソール］の順にクリックします。



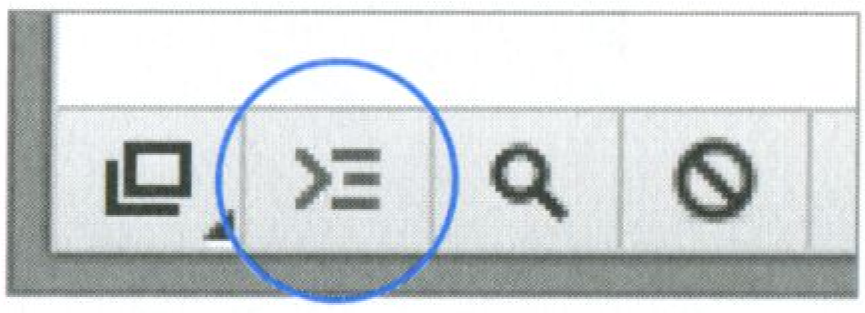
Chromeでコンソール画面を表示

［JavaScriptコンソール］をクリックすると、次のように下部にコンソールが表示されます。「Hello World!!」と値が表示されていることを確認してください。



Chromeのコンソール画面

コンソールが表示されない場合は、画面左下の右から2番目のアイコンをクリックしてください。



コンソールを表示するために毎回 ≡ をクリックしメニューをたどるのは面倒、という場合はショートカットキーを利用しましょう。Webページを表示した状態で、キーボードの［Ctrl］＋［Shift］＋［J］キーを押すと、コンソールを表示できます。

JavaScriptを外部ファイルに記述している場合も、同様にコンソールを利用できます。たとえば、以下は75ページで作成したsample.jsの内容を「console.log("Hello World!!");」に書き換え、ブラウザでWebページとコンソールを表示した画面です。

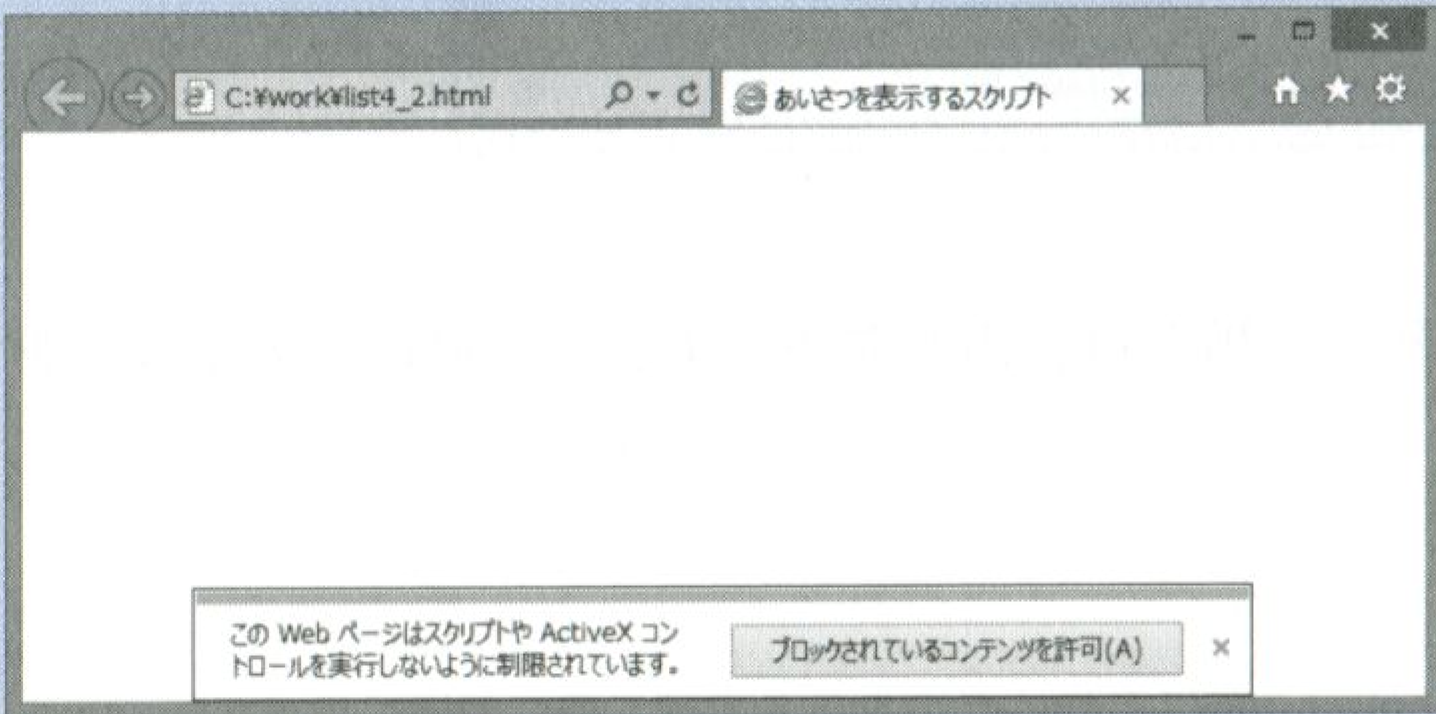


Column

Internet Explorer 9でWebページとコンソールを表示する場合

●Webページを表示するときの注意点

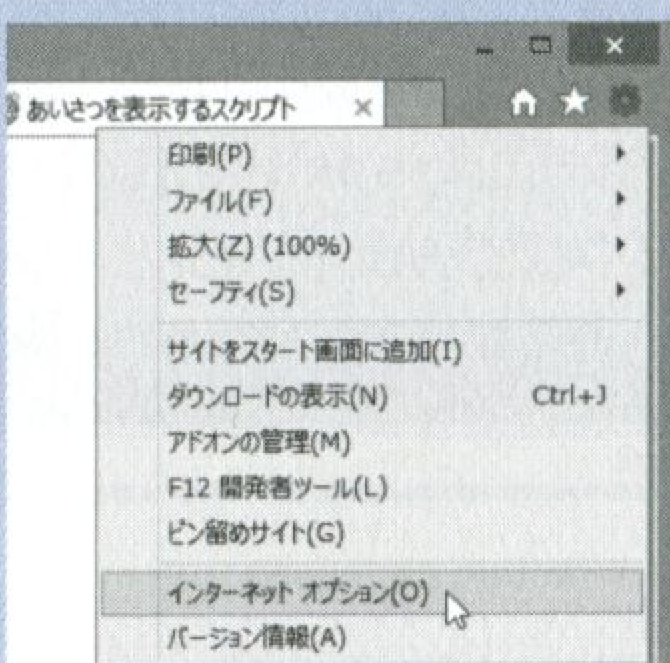
Internet ExplorerでローカルPC上に置かれたファイル（Webサーバーにアップロードされていないファイル）を実行する場合、次のような確認メッセージが表示されます。



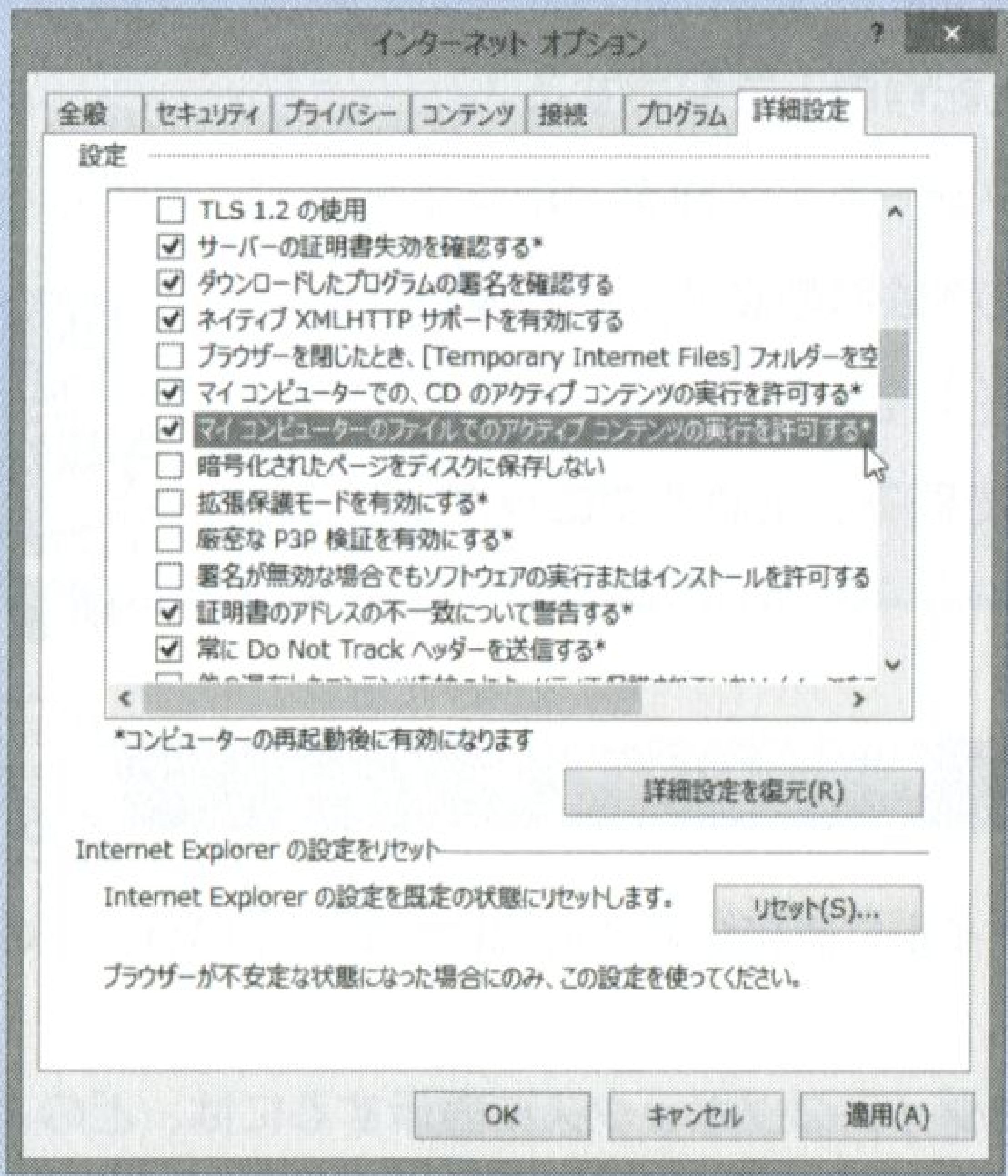
Internet Explorer 9の確認メッセージ

これはWebページ上でJavaScriptなどを実行する場合に表示されるメッセージです。JavaScriptを実行するには［ブロックされているコンテンツを許可(A)］をクリックしてください。今後このメッセージが表示されないようにするには、次の設定を行ないます。

❶ Internet Explorerの右上にある⚙をクリックして［インターネットオプション(O)］をクリック。



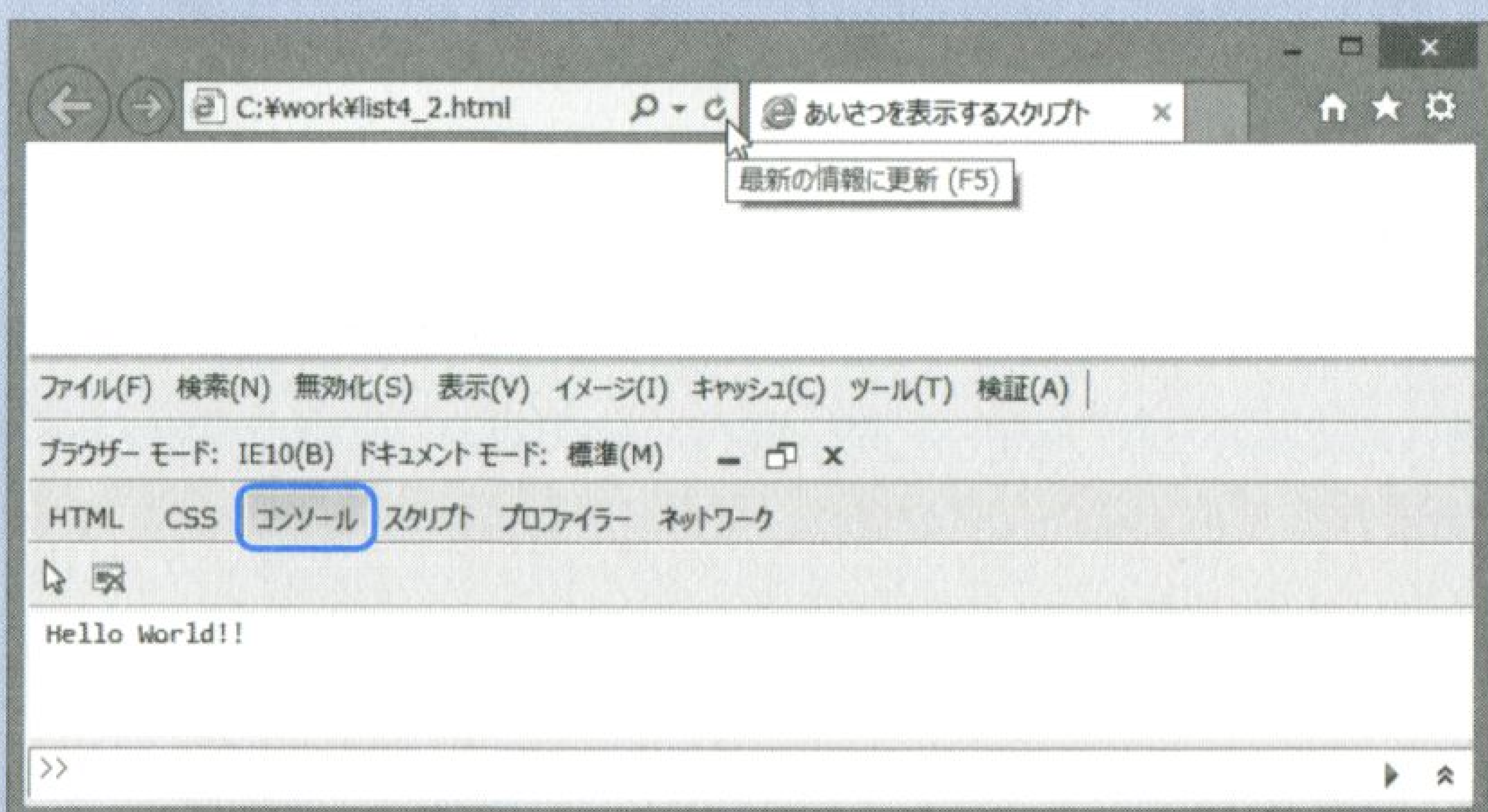
❷「詳細設定」タブで［マイコンピューターのファイルでのアクティブコンテンツの実行を許可する］にチェックを入れる。



以上で設定は完了です。

●コンソールの利用

Webページを表示した状態で、キーボードの［F12］キーを押すと、画面下に帯状の枠が表示されます。以下のように「コンソール」タブをクリックすると、コンソールが表示されます。コンソールに「Hello World!!」が表示されないときには、ブラウザ上部の をクリックしてページを最新の状態にしてください。



Point!

- JavaScriptは半角の英数字と記号のみで記述する。
- 文字列を扱う場合はシングルクォート(')かダブルクォート(")で囲む。
- JavaScriptでは大文字と小文字は別の文字として扱われる。
- 命令文の末尾にはセミコロン(;)を付ける。
- 複数行の命令群を波かっこ{ }で囲み、それを**ブロック**と呼ぶ。

■ Check Test 1

Q1 JavaScriptをHTMLファイルに直接記述する場合、なんというタグの中に記述しますか？

Q2 「こんにちは」という文字をダイアログボックスに表示するには、どのような命令を記述しますか？

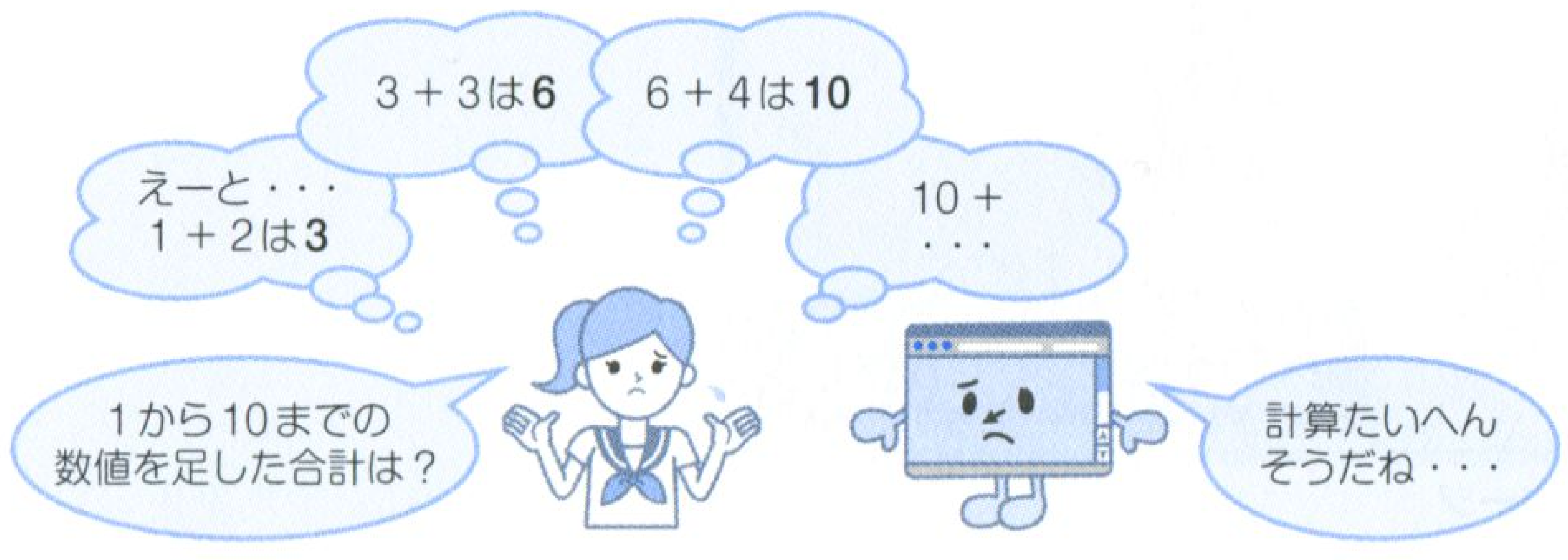
➲解答は巻末に掲載

4-2 変数とデータ型

変数とは

では、JavaScriptでスクリプトを記述するときに必要な知識を学んでいきましょう。パソコンでは、文字や数値、記号など、さまざま情報を**データ**として扱います。JavaScriptなど、プログラミング言語では、このデータを使っていろいろな処理を行ないます。

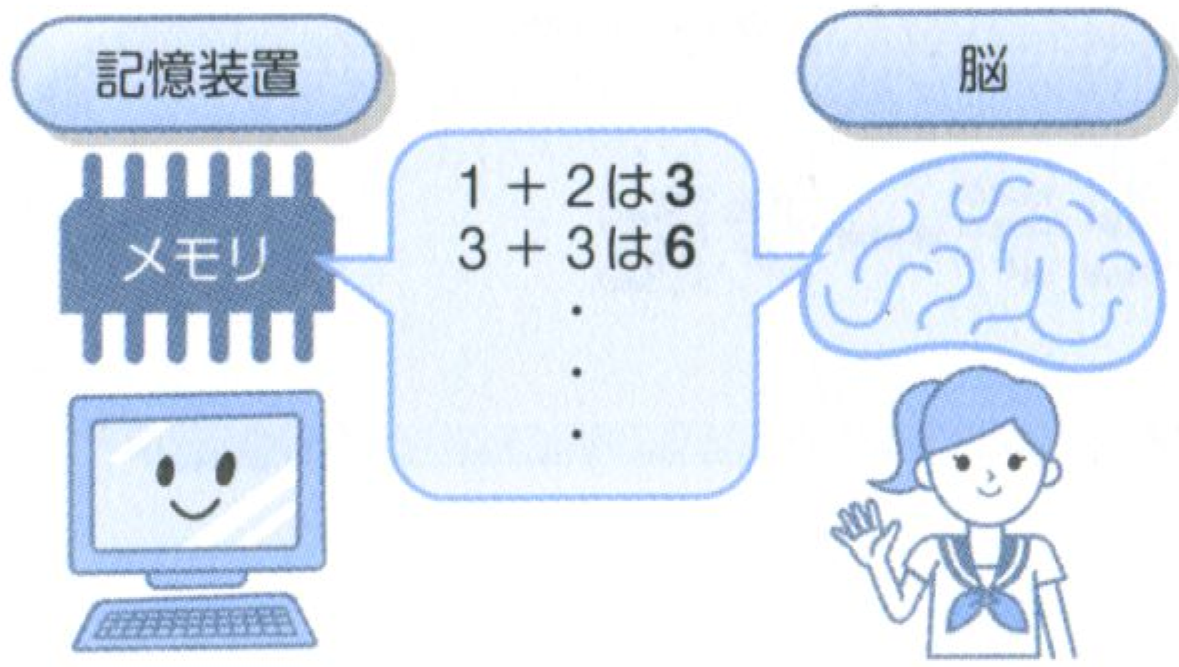
代表的な処理といえば計算です。たとえば、人間が1から10までの合計値を暗算で求める場合、頭の中でどのように計算するでしょうか？



複雑な暗算はたいへん！

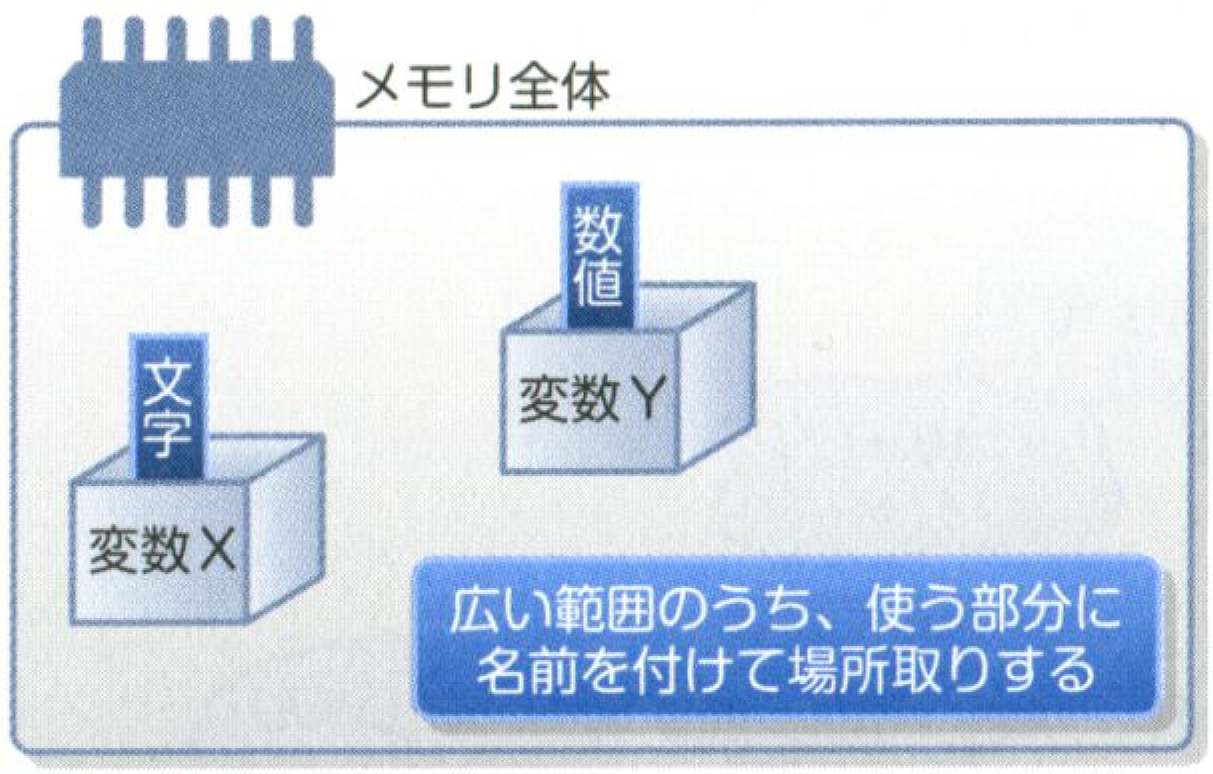
まず1＋2の結果を求め、その結果に3を足し、さらにその結果に4を足し・・・、というように、途中の計算結果をいったんおぼえて、そこに次の数値を加算していくでしょう。

JavaScriptで計算を行なう場合も同様に、途中の計算結果をどこかにおぼえておく必要があります。JavaScriptがデータをおぼえるとき、実際はパソコンの**メモリ**という装置にデータを記憶しています。



一時的に記憶する場所

メモリがデータを記憶できる領域はとても広いので、その中でデータを個別に管理するために、領域の一部を場所取りして名前を付けます。これが**変数**（へんすう）です。変数とは、メモリ上に作られたデータを入れるための小さな箱のようなものです。



変数のイメージ

変数の宣言

変数を利用するためには、まず**宣言**（せんげん）を行なう必要があります。宣言とは、変数に名前を付けて、利用のための準備をすることです。宣言は一度だけ行ないます。同名の変数を何度も宣言することはできません。

文法 変数宣言

```
var 変数名;
```

変数名の前に半角スペースを入れる

```
var 変数1, 変数2 ・・・;
```

複数の変数を宣言する場合はカンマで区切る

■記述例

```
var x;
```

```
var x, y, z;
```

注意 **変数宣言を省略すると通常の変数ではなくなる！**

実はJavaScriptでは、宣言をしなくても変数を利用できます。変数宣言を省略すると、通常の変数とは異なる種類の変数になります。詳しくは第6章で説明します。

◇ 変数の命名規則

変数には、自由な名前を付けることができます。ただし、JavaScriptの**予約語**と同じ名前にはできません。読みやすいスクリプトにするために、なんのデータを扱う変数なのかが推測しやすい名前にしましょう。たとえば、金額のデータを扱う変数であれば「money」や「price」などが良いでしょう。

用語解説 **【予約語】**

予約語とは、JavaScriptであらかじめ用意されている単語のことです。以下の予約語一覧の名前は変数名に使えないので気をつけましょう。

beak	do	if	switch	typeof	case
else	in	this	var	catch	false
instanceof	throw	void	continue	finally	new
true	while	default	for	null	try
with	delete	function	return		

また、以下の単語は将来的に予約語にされる可能性があるものです。これらも使用しないようにしましょう。

abstract	boolean	byte	char	class	const
debugger	double	enum	export	extends	final
float	goto	implements	import	int	interface
long	native	package	private	protected	public
short	static	super	synchronized	throws	transient
volatile					

注意 **大文字と小文字は別物！**

JavaScriptでは、大文字と小文字は異なる文字として認識されます。そのため、たとえば、変数「a」と変数「A」はまったく別の変数になります。

変数への値の代入

変数の中に値を入れることを**代入**（だいにゅう）と呼びます。変数が持てる値は1つだけなので、代入をするたびに値が上書きされます。

文法 **代入**

```
変数名 = 値;
```

注意 **算数の「=」とはちょっと違う！**

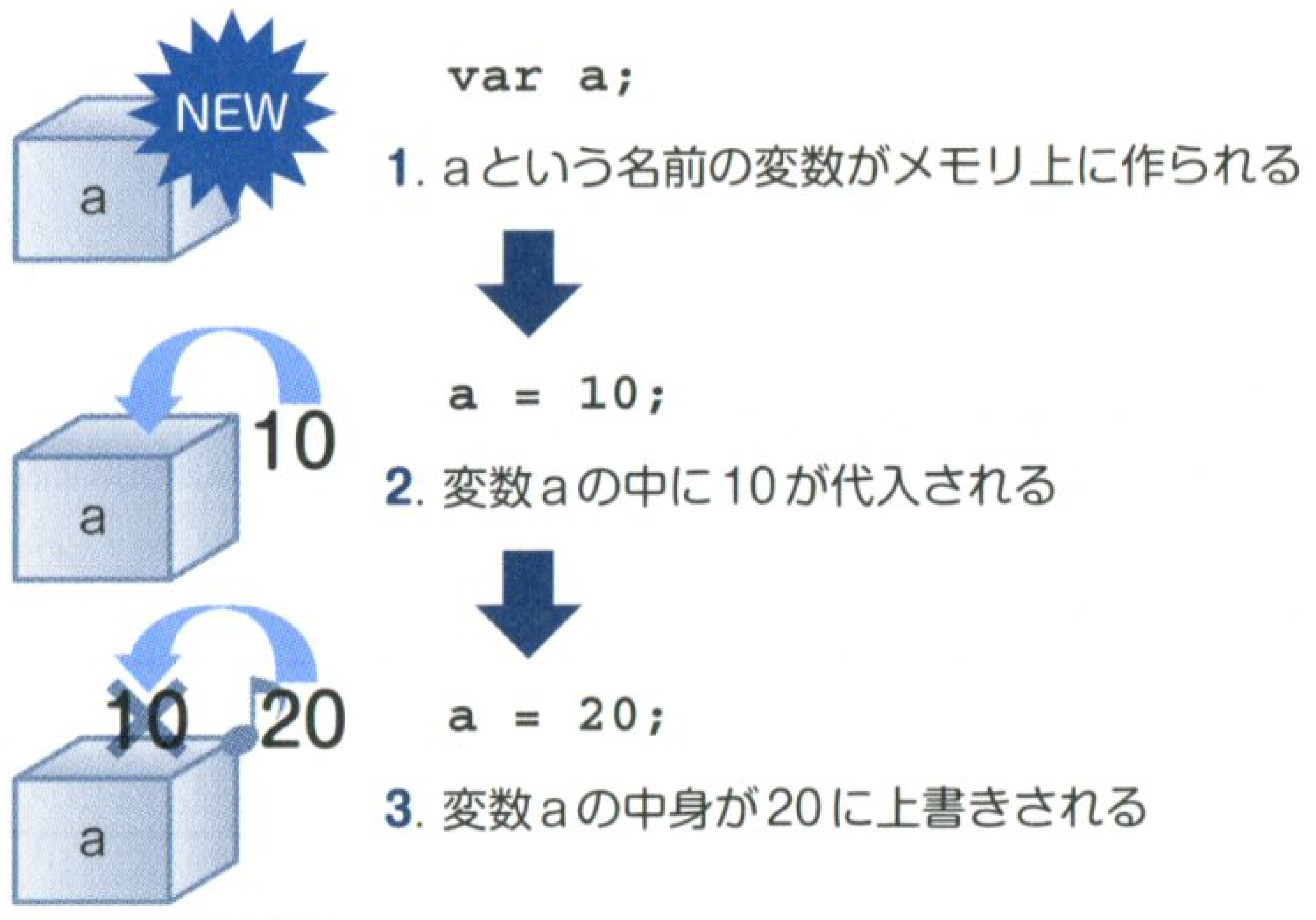
JavaScriptにおける=は、算数の=とは意味が異なります。JavaScriptの=は左辺と右辺が等しいという意味ではなく、「右辺を左辺に入れる」という意味です。

リスト4.3は、変数に対して数値を代入する例です。変数aに10を代入した後で20に上書きしているので、最終的にaの中に入っている値は20となります。

リスト4.3　変数への値の代入

```
var a;
a = 10;
a = 20;
```

JavaScriptは上から順番に実行されるため、この例が実行されると、変数aの内容が次のように変わっていきます。



変数への値の代入

◆ 変数の初期化

変数の宣言と代入は、同時に行なうこともできます。

文法 初期化

```
var 変数名 = 値;
```

■記述例

```
var x = 0;
```

このように宣言時に値を代入することを**初期化**（しょきか）と呼びます。変数宣言のみで初期化を行なわない場合、変数の中には初期値として**undefined**（アンディファインド）という特殊な値が入ります。undefinedに対して計算などの処理を行なうことはできないため、変数を計算などに利用する場合は必ず先に値を入れておきます。

変数の表示

変数の中身を表示するには、リスト4.4のように、console.logやalertの丸かっこ()内に変数名を指定します。実行時には変数の中身が表示されます。リスト4.5のように、変数名をダブルクォート（"）もしくはシングルクォート（'）で囲むと、囲まれた範囲は単純な文字列となり、変数名がそのまま表示されてしまいます。

リスト4.4　変数の中身の表示

```
var a = 10;
console.log(a);
```

■実行結果

```
10
```

リスト4.5　変数の中身の表示（誤った例）

```
var a = 10;
console.log("a");
```

■実行結果

```
a
```

> **注意　変数名は囲まない！**
>
> 変数の中身を表示したい場合は、変数名をダブルクォート（"）やシングルクォート（'）で囲んではいけません。

データ型

データ型とは、変数に入っているデータの種類のことです。たとえば、文字列と数値は、異なる種類のデータ型になります。データ型が異なると、実行できる処理内容も異なります。文字列型のデータに対しては文字列の検索や置換などの処理ができますし、数値型のデータは計算処理に利用できます。



データ型はさまざまな処理を可能にする

JavaScriptには、表4.1のようなデータ型があります。

表4.1 JavaScriptのデータ型

データ型	説明	例
数値	整数や実数	10 0.5
文字列	シングルクォート（'）またはダブルクォート（"）で囲まれた文字の並び。全角文字も扱うことができる	'abc' "こんにちは"
論理値	以下の２つのうちどちらかの値 ・真（条件を満たす）、正しい（等しい）、はい、YES 等の意味を持つ「true」 ・偽（条件を満たさない）、正しくない（等しくない）、いいえ、NO 等の意味を持つ「false」	true false
配列	複数のデータの集合体	[10, 20, 30]
連想配列 （オブジェクト）	キーと値の集合体	{ width : 90, height : 50 }
null 値	オブジェクト（第７章）が存在しないことを表わす特殊なデータ型	null
undefined 値	変数が初期化されていない、あるいはプロパティ（第７章）が定義されていないことを表わす特殊なデータ型	undefined
NaN 値	数値でないことを表わす特殊なデータ型	NaN

JavaScriptはデータ型について厳密なチェックを行なわないため、数値型の値が入っていた変数を文字列型の値で上書きする、といった処理も問題なく実行できます。

Point!

- ▶ スクリプト中で扱うデータは変数に入れる。
- ▶ 変数に入れるデータの種類のことをデータ型と呼ぶ。



■ Check Test 2

Q1 以下は、xという名前の変数を宣言する文です。空欄にはなにが入りますか？

```
[      ] x;
```

Q2 変数xの中身をコンソールに表示するには、どのような命令を記述しますか？

➲ 解答は巻末に掲載

4-3 演算子

演算子とは

演算子とは、なんらかの処理を行なう記号のことです。JavaScriptには、さまざまな演算子がありますが、大きく分けて**算術演算子**、**代入演算子**、**比較演算子**、**論理演算子**の4種類があります。

```
var a = 10 + 20;

alert ( a < 50 );

alert ( a < 50 && a > 20 );
```

- `+`：算術演算子（四則演算を行なう）
- `=`：代入演算子（変数に値を代入する）
- `<`：比較演算子（2つの値を比較する）
- `&&`：論理演算子（2つの条件式をつなげる）

演算子の種類

ここではまず、基本的な演算子である算術演算子、代入演算子の2つについて学び、残る比較演算子、論理演算子については第5章であらためて取り上げます。

算術演算子

算術演算子は、計算を行なう演算子です（表4.2）。

表4.2 JavaScriptの算術演算子

演算子	説明	使用例	結果
+	数値の加算	1 + 1	2
+	文字列の連結	"Hello" + "World"	"HelloWorld"
-	数値の減算	2 - 1	1
*	数値の乗算	2 * 2	4
/	数値の除算	10 / 2	5
%	数値の剰余（除算の余り）	10 % 3	1

＋演算子は、計算対象のデータ型に応じて処理内容が変わります。どちらか片方でも文字列であれば、結果は2つの値を連結した文字列となります。

たとえば、リスト4.6のⒶとⒷは、一見同じように見える計算処理ですが、データ型の違いによってまったく異なる実行結果になります。

リスト4.6　＋演算子の挙動の違い

```
var x = 123 + 456;———————Ⓐ
var y = "123" + "456";———Ⓑ
console.log(x);
console.log(y);
```

■実行結果

```
579 ———— Ⓐの結果
123456 ———— Ⓑの結果
```

また、「%」演算子による剰余算は、「ある数値で割り切れるかどうか」を調べる際によく使われます。たとえば、「x % 2」という計算式を実行すると、xが偶数であれば0、奇数であれば1が求められます。

複合代入演算子

複合（ふくごう）代入演算子は、変数に対して計算と代入を同時に行なう演算子です（表4.3）。

表4.3　複合代入演算子

演算子	説明	使用例	結果（a の値）
+=	左辺の値に右辺の値を加算したものを代入	a = 1; a += 2;	3
-=	左辺の値から右辺の値を減算したものを代入	a = 5; a -= 2;	3
*=	左辺の値に右辺の値を乗算したものを代入	a = 3; a *= 2;	6
/=	左辺の値を右辺の値で除算したものを代入	a = 10; a /= 2;	5
%=	左辺の値を右辺の値で除算した余りを代入	a = 10; a %= 3;	1
++	変数に 1 加算する（インクリメント）	a = 1; a++;	2
--	変数から 1 減算する（デクリメント）	a = 1; a--;	0

用語解説

【インクリメント、デクリメント】

インクリメントとは変数の値を1増やすこと、デクリメントとは変数の値を1減らすことです。++はインクリメント演算子、--はデクリメント演算子とも呼びます。

複合代入演算子は、「変数 = 変数 + 値」という形の式を省略するための演算子です。リスト4.7とリスト4.8の演算子は、いずれもまったく同じ処理を行なっています。

リスト4.7　+演算子による加算

```
var a = 10;
a = a + 5;      // (a+5) は (10 + 5) なので、aの値が15で上書きされる
console.log(a);
```

リスト4.8　+=演算子による加算

```
var a = 10;
a += 5;         // aの値に5加算されて15になる
console.log(a);
```

■実行結果　リスト4.7・リスト4.8

```
15
```

また、リスト4.9からリスト4.11の演算子は、いずれも同じ処理を行っています。

リスト4.9　+演算子によるインクリメント

```
var a = 10;
a = a + 1;      // (a+1) は (10 + 1) なので、aの値が11で上書きされる
console.log(a);
```

リスト4.10　+=演算子によるインクリメント

```
var a = 10;
a += 1;         // aの値に1加算されて11になる
console.log(a);
```

リスト4.11　++演算子によるインクリメント

```
var a = 10;
a++;          // aの値に1加算されて11になる
console.log(a);
```

■実行結果　リスト4.9・リスト4.10・リスト4.11

```
11
```

Point!

▶ 演算でデータの計算や代入、比較などの処理を行なう。



■ Check Test 3

Q1 以下のスクリプトを実行した結果、変数xに代入される値はなんですか？

```
var x = "100" + 100;
```

Q2 以下のスクリプトを実行した結果、最終的に変数yに代入される値はなんですか？

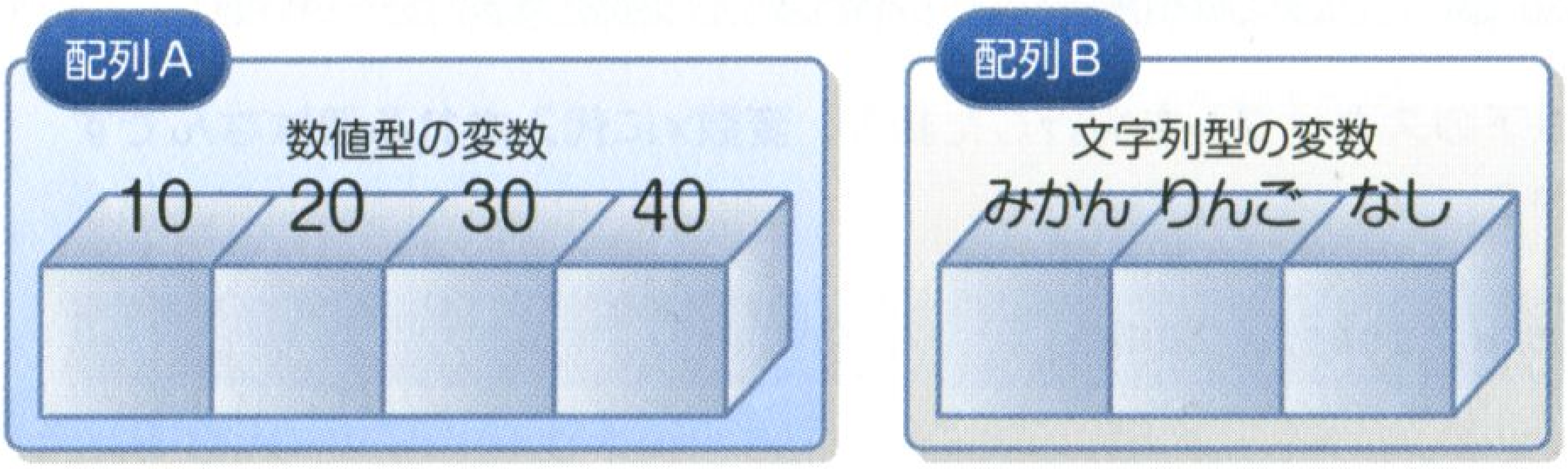
```
var y = 10;
y *= 3;
y++;
```

➲解答は巻末に掲載

4 配列

配列とは

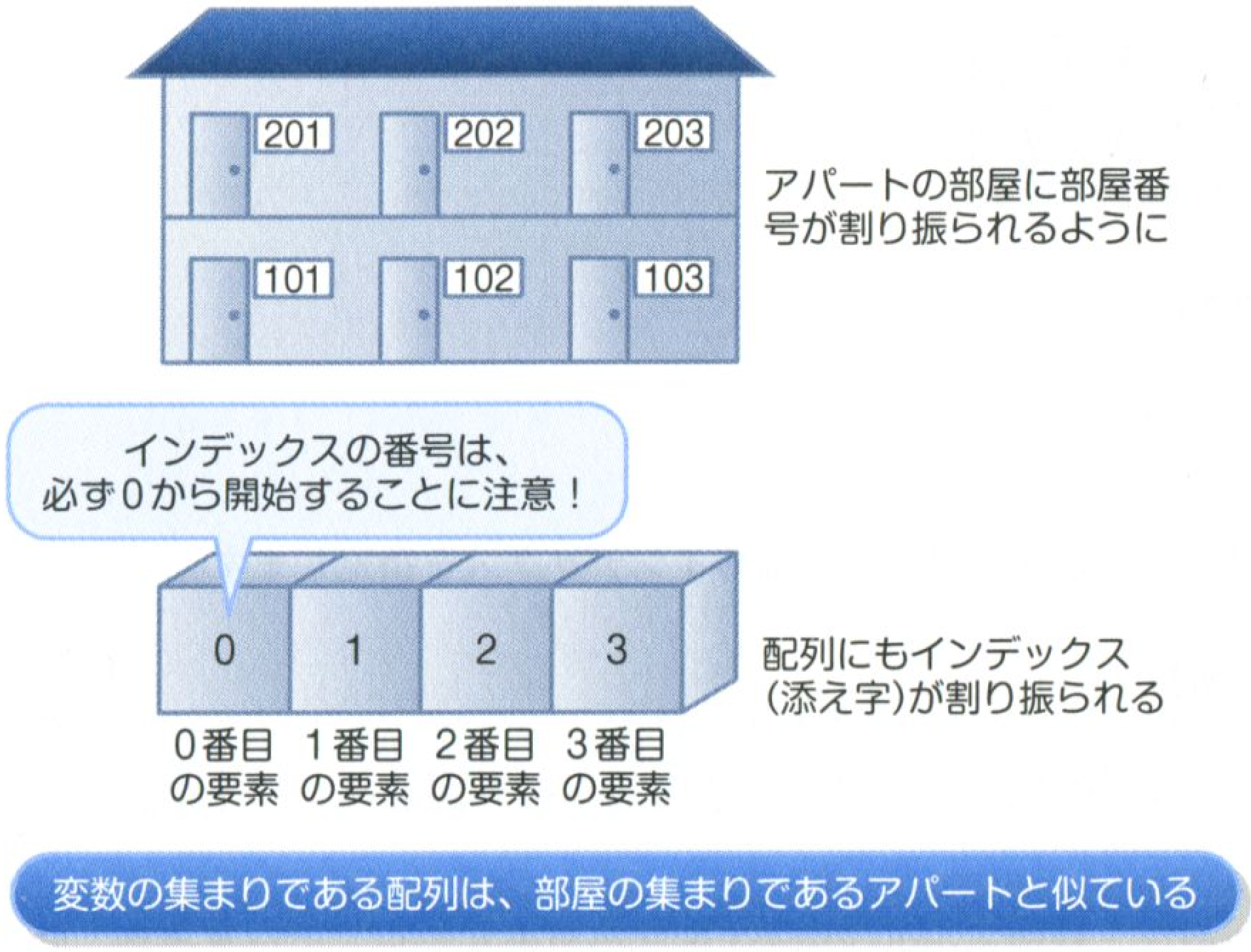
4.2節で、変数に入れることのできる値は1つだけだということを学びました。では、100件のデータをおぼえておきたいときは変数を100個宣言しなければならないのでしょうか？　考えるだけでうんざりしてしまいますね。このようなときに利用するのが**配列**です。配列とは、同じ用途で用いる複数の変数を1つにまとめたもののことです。



配列は変数の集まり

配列は要素の集合体

配列は変数の集まりによって構成されます。個々の変数のことを**要素**と呼びます。各要素を識別するために、**インデックス**と呼ばれる番号（0から始まる整数値）が割り振られます。インデックスは**添え字**とも呼ばれます。



配列のイメージ

配列の書き方

◆ 配列の宣言

配列の宣言は、変数と同じようにvarを使います。角かっこ[ ]内には、要素に代入する値をカンマ区切りで指定します。

文法 配列の宣言と初期化

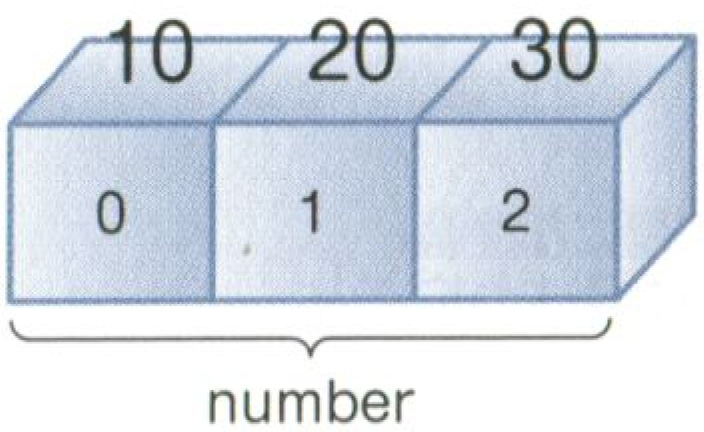
```
var 配列名 = [ 値1, 値2, 値3 ・・・ ];
```

配列は角かっこ[ ]で要素を囲む

リスト4.12のように宣言と初期化を行なうと、要素3つ分の配列が作られます。

リスト4.12 配列の初期化

```
var number = [ 10, 20, 30 ];
```



配列のイメージ

◆ 要素の参照

配列内の要素を参照するには、要素のインデックスを指定します。

文法 要素の参照

```
配列名[インデックス];
```

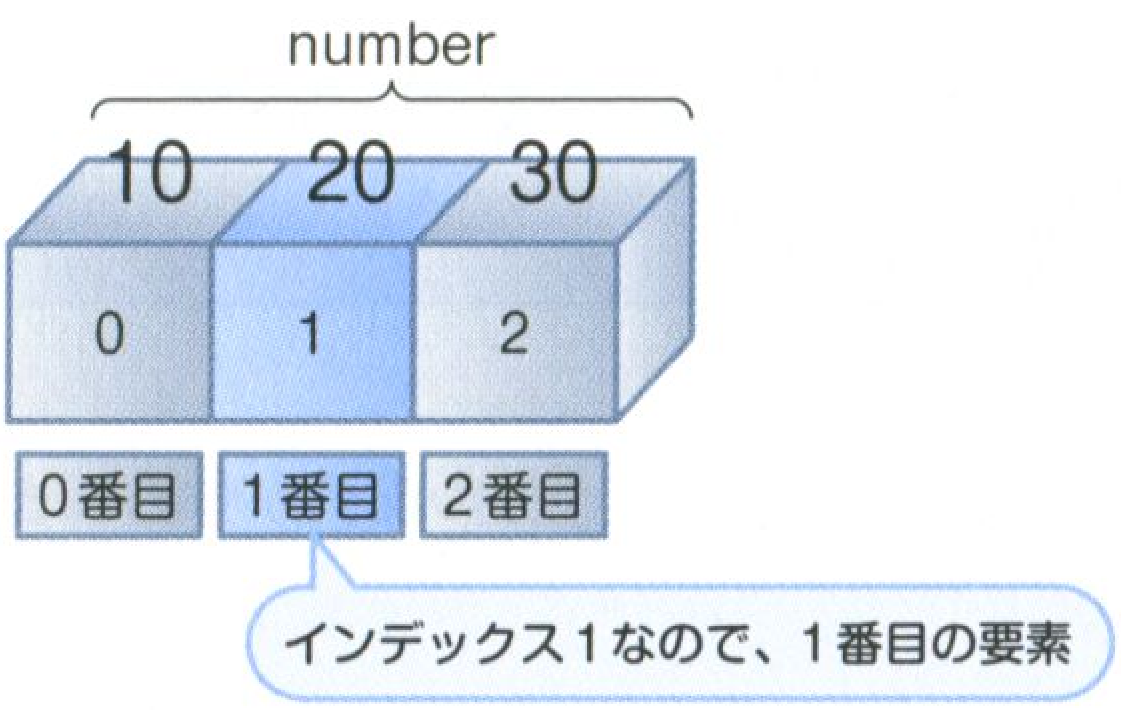
リスト4.13では、配列number内の1番目の要素を指定しているので、20が表示されます。

リスト4.13 配列要素の参照

```
var number = [ 10, 20, 30 ];
console.log(number[1]);
```

■実行結果

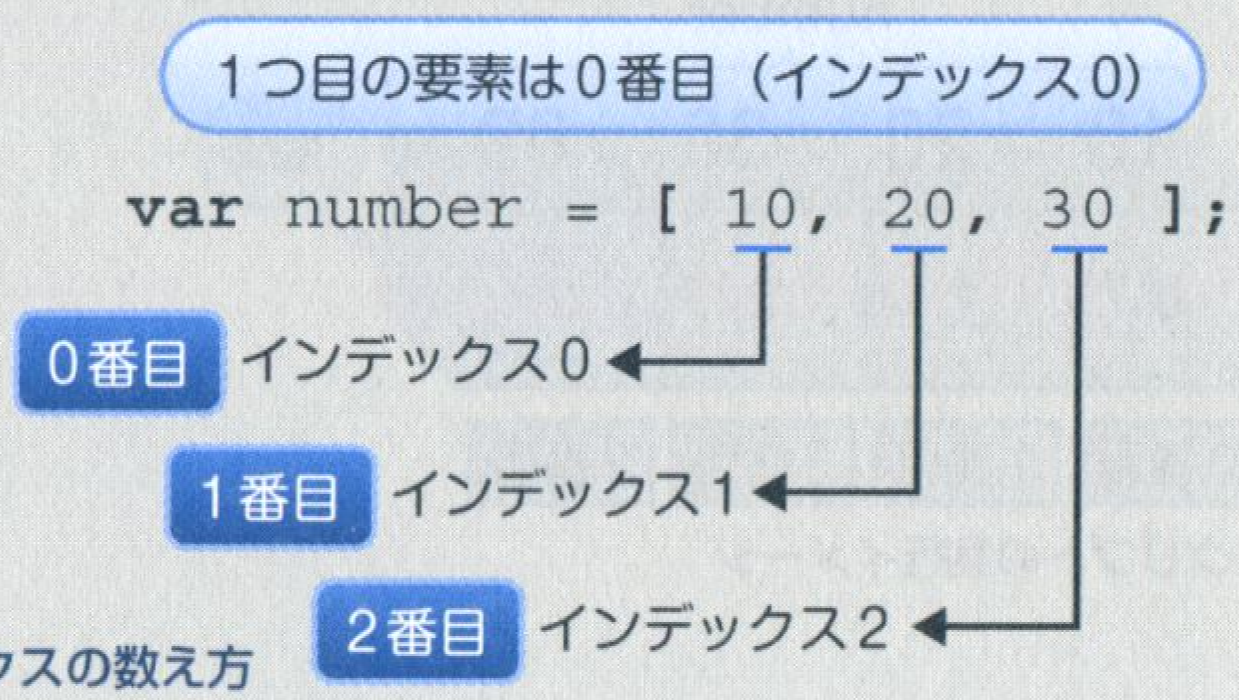
```
20
```



スクリプトの実行イメージ

注意 インデックスは0から始まる！

インデックスは0から開始する数値です。1からの開始ではないので気をつけましょう。初めてだとわかりにくいのですが、1つ目の要素は0番目（=インデックス0)、2つ目の要素は1番目（インデックス1)・・・というように、通常の数え方と異なる点に注意してください。



インデックスの数え方

◆ 要素への代入

各要素に対して、個別に値を代入できます。

文法 要素への代入

```
配列名[インデックス] = 値;
```

リスト4.14では、1番目の要素の値を上書きしています。

リスト4.14 配列要素への代入

```
var number = [ 10, 20, 30 ];
number[1] = 50;
console.log(number[1]);
```

■実行結果

```
50
```

◆ 要素の追加

配列内に存在しないインデックスを指定することで、要素を追加することもできます。リスト4.15は、インデックスに3を指定することで、3番目の要素を新たに生成しています。

リスト4.15　インデックスの指定

```
var number = [ 10, 20, 30 ];
number[3] = 99;
```

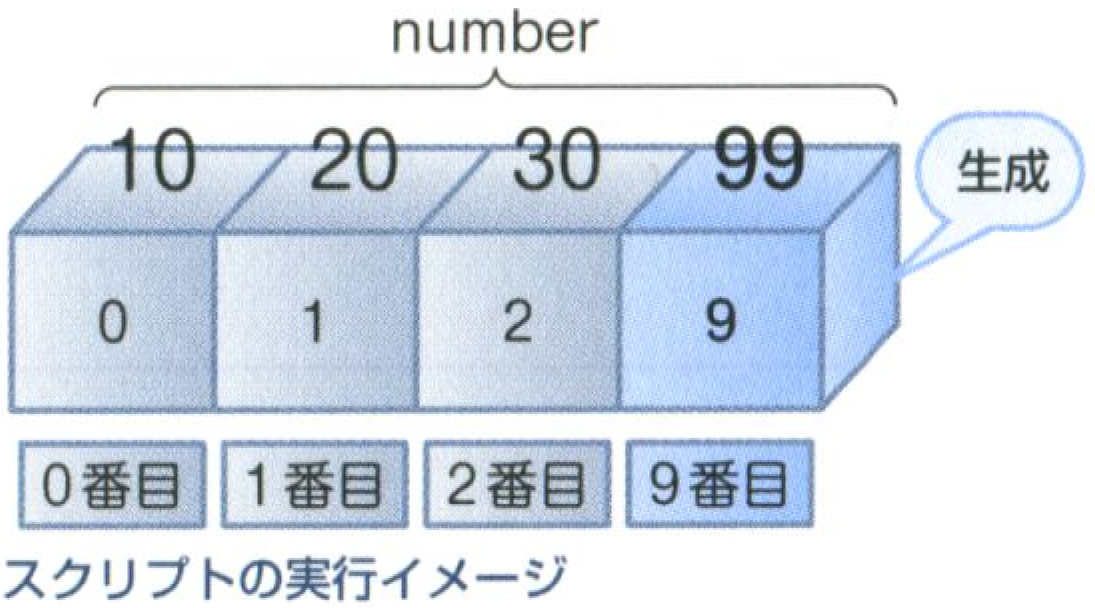


スクリプトの実行イメージ

要素を追加するとき、インデックスは連続した番号でなくてもかまいません。リスト4.16のように離れたインデックスを指定すると、途中の番号が欠番となります。

リスト4.16　離れたインデックスの指定

```
var number = [ 10, 20, 30 ];
number[9] = 99;
```



スクリプトの実行イメージ

配列の要素に指定する値が最初から決まっていない場合は、まず空(から)の配列を作っておいて、後から要素を追加しても良いでしょう。リスト4.17では、リスト4.12とまったく同じ配列を生成しています。

リスト4.17　空の配列の作成

```
var number = [ ];          // 空の配列の作成
number[0] = 10;
number[1] = 20;
number[2] = 30;
```

◆ 要素数の取得

配列名の後ろに.lengthを付けると、配列が保持する要素の数を表わします。

文法 要素数の取得

```
配列名.length
```

厳密には最大インデックス+1を表わすので、リスト4.16のようにインデックスに欠番がある場合は、要素数ではなくなります。

リスト4.18 配列の要素数の取得

```
var number = [ 10, 20, 30 ];
console.log(number.length);
```

■実行結果

```
3
```

二次元配列

配列の集合体を**二次元配列**と呼びます。リスト4.19は、2つの配列を要素とした二次元配列です。小さい配列が大きい配列の中に入っている構造をイメージしてください。

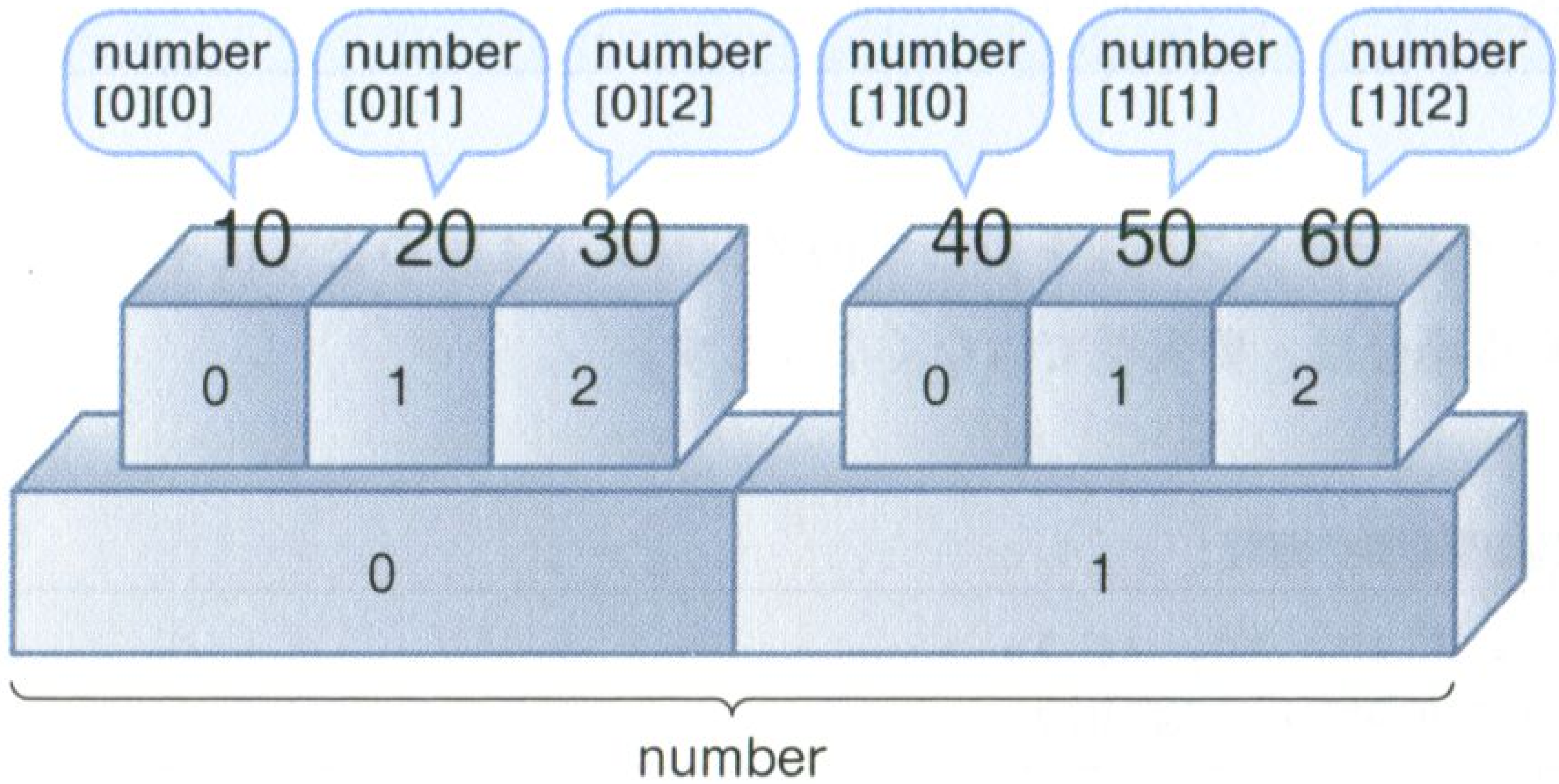
リスト4.19 二次元配列

```
var number = [ [ 10 , 20 , 30 ] , [ 40 , 50 , 60 ] ];
console.log(number[0][0]);
console.log(number[0][1]);
console.log(number[0][2]);
console.log(number[1][0]);
console.log(number[1][1]);
console.log(number[1][2]);
```

■実行結果

```
10
20
```

```
30
40
50
60
```



二次元配列のイメージ

さらに角かっこ[ ]を入れ子にすることで、三次元、四次元の配列を作ることもできます。しかし、あまり階層が深くなると読みづらいスクリプトになってしまうため、二次元程度におさえておくのが現実的でしょう。

Point!

- ▶ 似たようなデータを大量に扱う場合は配列に入れる。
- ▶ 配列の初期値は角かっこ [ ] で囲む。
- ▶ 配列の要素にはインデックス（添え字）と呼ばれる識別番号が付けられる。



■ Check Test 4

Q1 以下のスクリプトを実行した結果、コンソールに表示される値はなんですか？

```
var fruits = ["りんご", "みかん", "パイナップル", "ぶどう", "バナナ" ];
console.log(fruits[2]);
```

Q2 以下のスクリプトを実行した結果、コンソールに表示される値は何ですか？

```
var fruits = ["りんご", "みかん", "パイナップル", "ぶどう", "バナナ" ];
console.log(fruits.length);
```

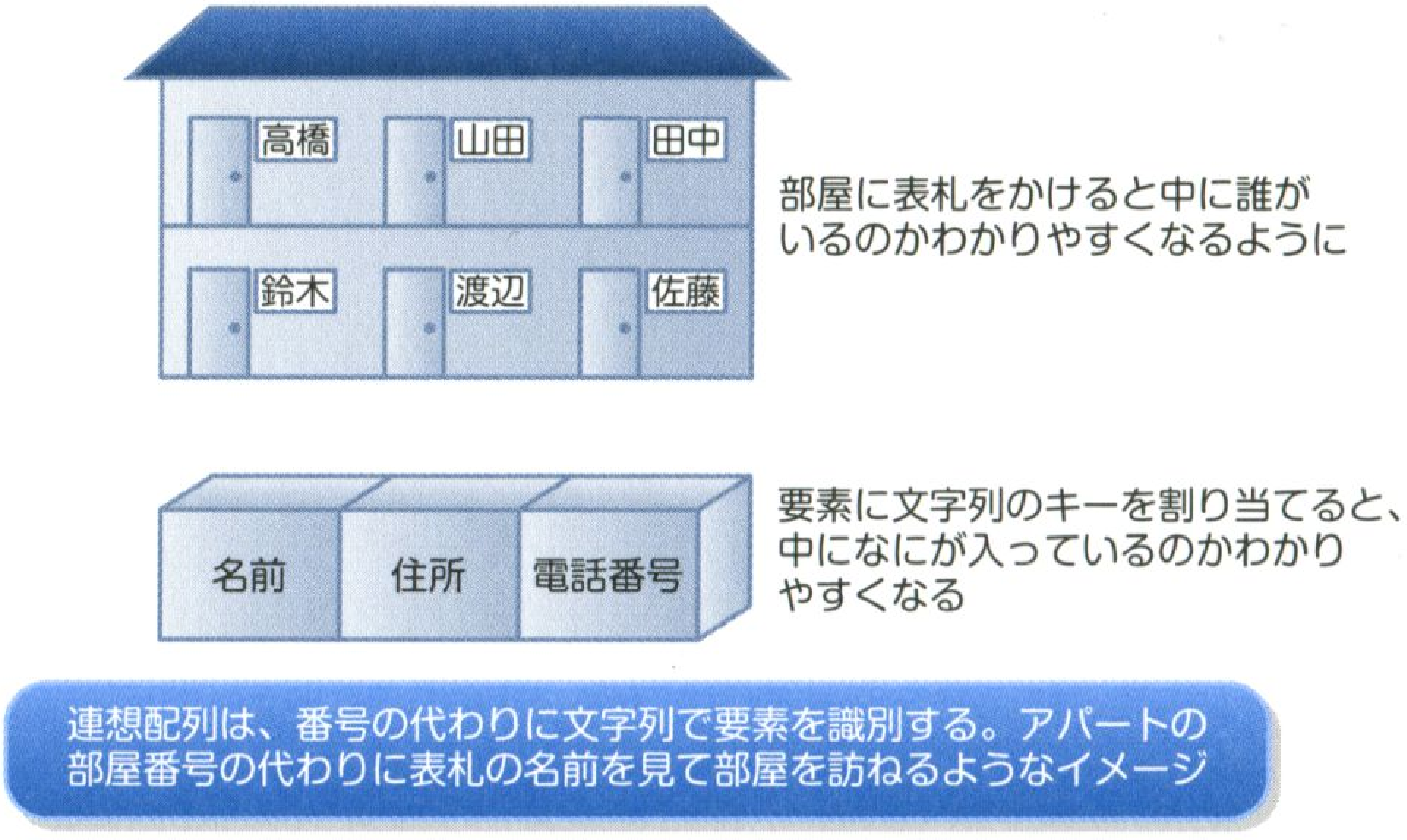
➲解答は巻末に掲載

4-5 連想配列

連想配列とは

配列とは、複数のデータの集合体のことでした。仮に、個人情報を管理する配列を作成し、0番目には名前、1番目には住所、2番目には電話番号を入れておいたとします。スクリプト量が多くなると、いざ配列要素の値を参照するとなったときに「住所を入れておいたのは何番目だっけ？」といった混乱が起こりがちです。

そんなときに利用するのが**連想配列**（れんそう）です。連想配列は、数値の代わりに意味のある文字列をインデックスとすることで、各要素を識別しやすくすることができます。連想配列の場合、インデックスのことを**キー**と呼びます。



連想配列のイメージ

◆ 連想配列の宣言と初期化

連想配列の要素は、キーと値のセットで1つとして扱われます。全体を囲むのは、角かっこ[ ]ではなく、波かっこ{ }であることに注意してください。

文法 連想配列の宣言と初期化

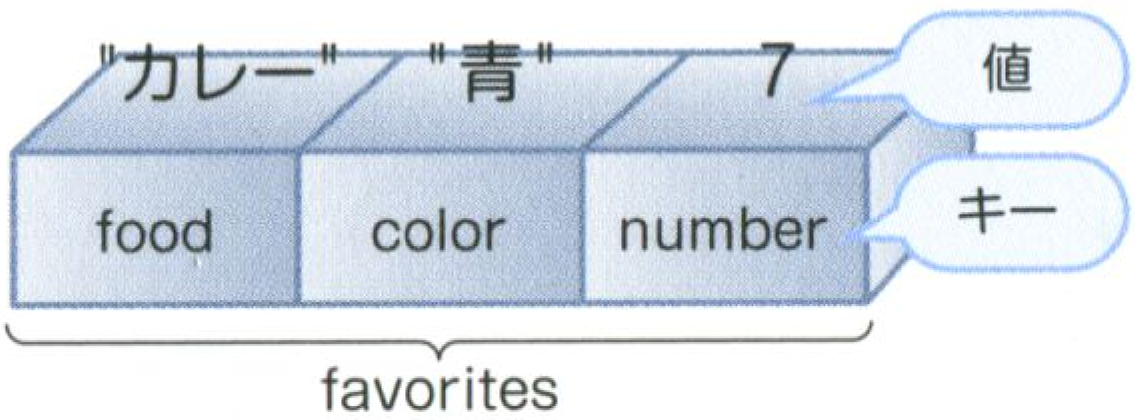
```
var 連想配列名 = { キー1 : 値1, キー2 : 値2, キー3 : 値3 ・・・ };
```

連想配列は波かっこ{ }で囲み、キーと値はコロン:で区切る

リスト4.20では、3つ分の要素を持つ連想配列を作成しています。

リスト4.20 連想配列

```
var favorites = {
    food : "カレー",
    color : "青",
    number : 7
};
```



スクリプトの実行イメージ

初期化時にキーをダブルクォートもしくはシングルクォートで囲むと、全角文字も使えるようになります。

リスト4.21 連想配列の初期化

```
var favorites = {
    "食べ物" : "カレー",
    "色" : "青",
    "数字" : 7
};
```

◆ 要素の参照

連想配列内の要素を参照するには、ドット(.)または[ ]でキーを指定します。

文法 要素の参照

```
連想配列名.キー
```

または

```
連想配列名["キー"]
```

※シングルクォートでも可

上記はどちらも同じ処理を行ないますが、角かっこ[ ]を使う方法の場合、キーを変数に入れて扱うことができます。

リスト4.22 ドット(.)を使った要素の参照

```
var favorites = {
    food : "カレー",
    color : "青",
    number : 7
};
console.log(favorites.food);
console.log(favorites.color);
console.log(favorites.number);
```

■実行結果

```
カレー
青
7
```

リスト4.23 角かっこ[ ]を使った要素の参照

```
var favorites = {
    food : "カレー",
    color : "青",
    number : 7
};
console.log(favorites["food"]);
var key = "food";          // キーを一度変数に入れる
console.log(favorites[key]);
```

■実行結果

```
カレー
カレー
```

◆ 要素への代入

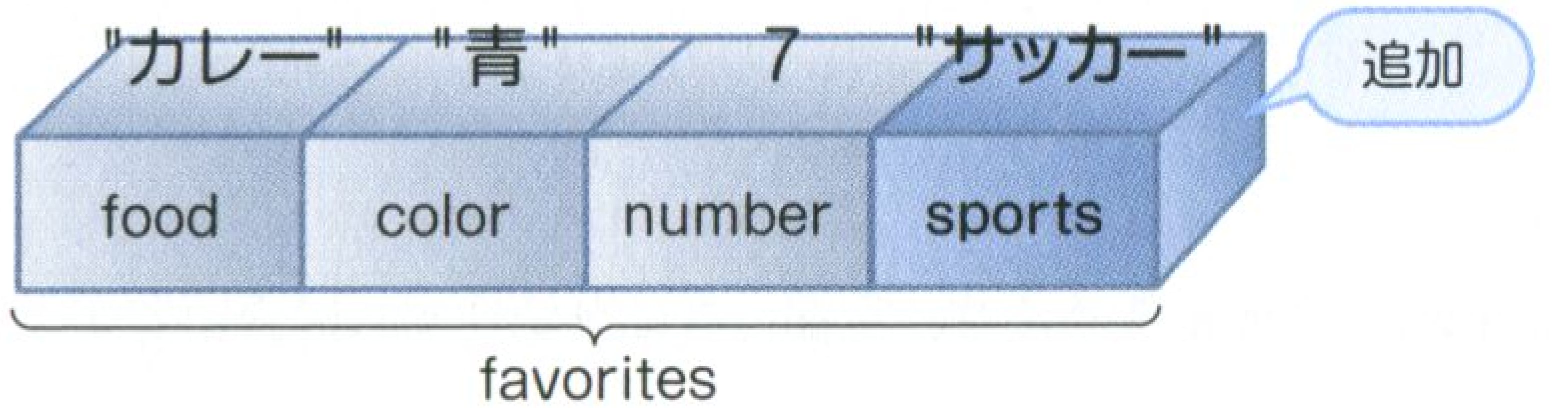
連想配列も配列と同様に、要素が持つ値を上書きできます。また、連想配列内に存在しないキーを指定することで、要素を追加できます。

リスト4.24　連想配列への要素の追加

```
var favorites = {
    food : "カレー",
    color : "青",
    number : 7
};
favorites.sports = "サッカー";     // 要素が追加される
console.log(favorites.sports);
```

■実行結果

```
サッカー
```



スクリプトの実行イメージ

Point!



- ▶ 連想配列の初期値は波かっこ{ }で囲む。
- ▶ 連想配列の要素にはキー（要素名）を付ける。

この章の到達度チェック

Q1 同一ディレクトリ内のJavaScriptファイル「sample.js」をHTMLファイルに読み込むには、<script>タグをどのように記述すれば良いですか？

Q2 以下のスクリプトを実行した結果、コンソールに表示される値はなんですか？

```
var x = 10, y = 20;
x = y;
y = x;
console.log(y);
```

Q3 以下のスクリプトを実行した結果、コンソールに表示される値はなんですか？

```
var x = 10;
x += "10";
console.log(x);
```

Q4 コンソールに「50」と表示するには、空欄になにを記述すれば良いですか？

```
var number = [ [10, 20], [30, 40, 50, 60], [70, 80, 90] ];
console.log(number[      ]);
```

Q5 連想配列では、インデックスのことをなんと呼びますか？

Q6 以下は、コンソールに「山田太郎」と表示するスクリプトです。空欄にはなにが入りますか？

```
var user = { name : "山田太郎", age : 25, height : 170 };
console.log(user.[      ]);
```

Q7 以下は連想配列の宣言と初期化を行なうスクリプトですが、一部間違っている箇所があります。どこでしょうか？

```
var profile = [ height : 170, weight : 60 ];
```

➲解答は巻末に掲載

第5章

制御構文

この章で学ぶこと

- if文
- switch文
- while文
- for文
- for-in文
- 繰り返しの制御



● スクリプトには3通りの進み方がある ●

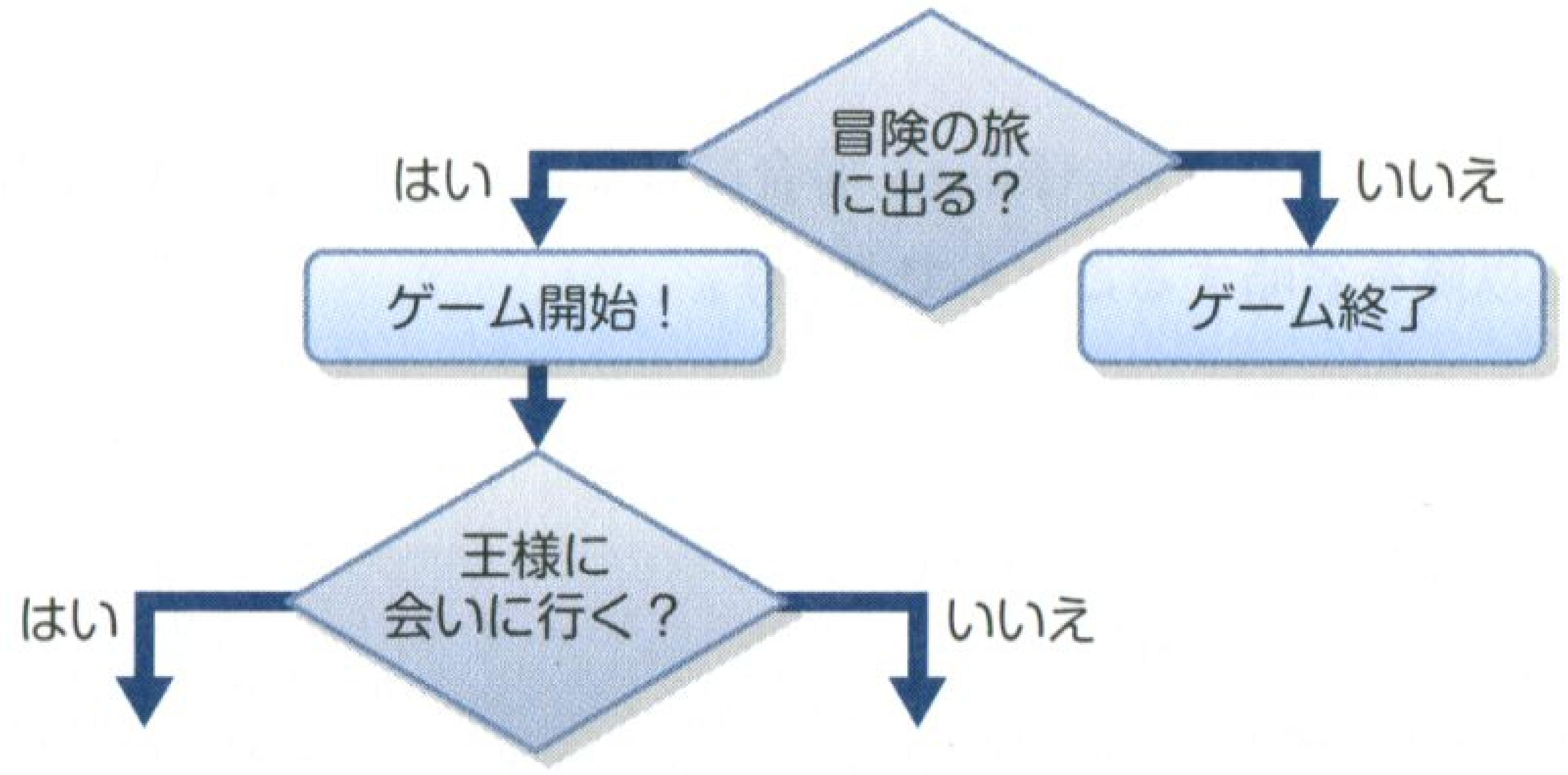


「順次」はスクリプトを単純に上から順に実行することで、第4章で学んだスクリプトはすべてこの「順次」でした。本章では、残りの2つである制御構文について学びましょう。

5 1 if文

分岐処理とは

ゲームなどではよく、選択肢に応じてその後のシナリオが変化する、といった仕組みがあります。そのように、ある条件に応じてその後行なわれる処理を振り分けることを**分岐**と呼びます。



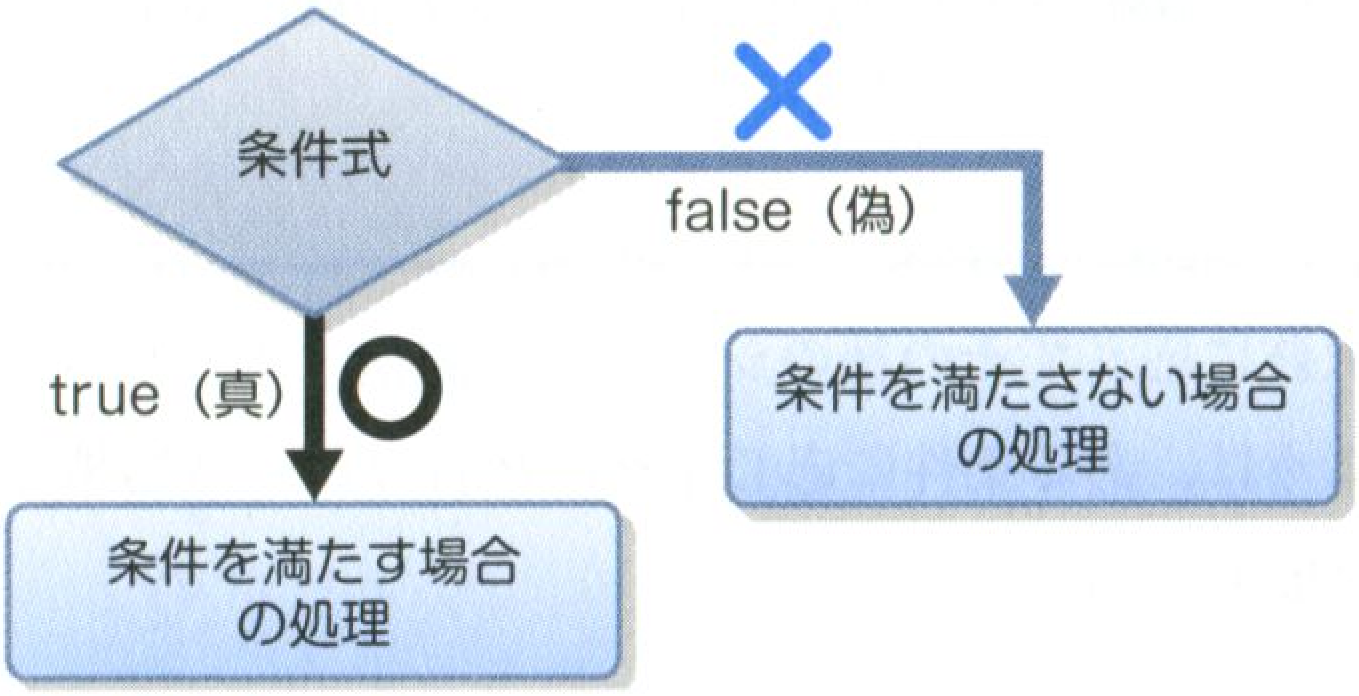
分岐処理のイメージ

JavaScriptでは、分岐処理を実現するための**構文**（書式）として**if文**が用意されています。if文を利用することで、ユーザーがどのオプションボタン（ラジオボタン）を選択したかを判定し、表示されるコンテンツを切り替えるWebページなどを作成できます。

文法 if文

```
if (条件式) {
    条件を満たす場合に実行する処理
} else {
    条件を満たさない場合に実行する処理
}
```

条件式とは、条件を指定し、かつその判定結果を論理値（true/falseのいずれか）で返す式のことです。この条件式で指定した条件を満たす（true）場合と、満たさない（false）場合とで異なる処理が実行されます



if文の処理イメージ

なお、条件式の条件を満たさない（false）場合にはなにもしない、という場合はelseキーワード以降を省略することもできます。

■記述例

```
if (条件式) {
    console.log("条件を満たす");
} else {
    console.log("条件を満たさない");
}
```

trueの場合

falseの場合

※　の部分は省略可能

条件式の書き方

条件式は、**比較演算子**を使って記述します。

比較演算子

演算子	説明	使用例	結果
==	左辺と右辺が等しい場合はtrue	`1 == 1`	`true`
!=	左辺と右辺が等しくない場合はtrue	`1 != 2`	`true`
<	左辺が右辺より小さい場合はtrue	`1 < 1`	`false`
<=	左辺が右辺以下の場合はtrue	`1 <= 1`	`true`
>	左辺が右辺より大きい場合はtrue	`1 > 1`	`false`
>=	左辺が右辺以上の場合はtrue	`1 >= 1`	`true`
===	左辺と右辺が等しく、かつデータ型も等しい場合はtrue	`1 === '1'`	`false`
!==	左辺と右辺が等しくない、もしくはデータ型が等しくない場合はtrue	`1 !== '1'`	`true`

リスト5.1は、信号機の色の状態を判定し、青であれば進み、それ以外の色であれば止まるように処理を分岐するスクリプトです。

リスト5.1　if文の利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   var color = "青";
   if (color == "青") {
       console.log("進む");
   } else {
       console.log("止まる");
   }
  </script>
 </head>
 <body>
 </body>
</html>
```

■実行結果

```
進む
```

変数colorに代入されている値は、

```
color == "青"
```

のように"青"なので、以下の条件式の結果はtrueとなります。

```
var color = "青";

if (color == "青") {
    console.log("進む");
} else {
    console.log("止まる");
}
```

trueなのですぐ下のブロックを実行

次のように、1行目で変数colorに別の値を代入すると、実行結果も変わります。

```
var color = "赤";
```

■実行結果

```
止まる
```

注意

=と==は別物

2つの値を比較する場合は、イコールを2つ（==）つなげて記述します。イコール1つだけ（=）にすると、右辺の値が左辺の変数に代入されてしまうので気をつけましょう。

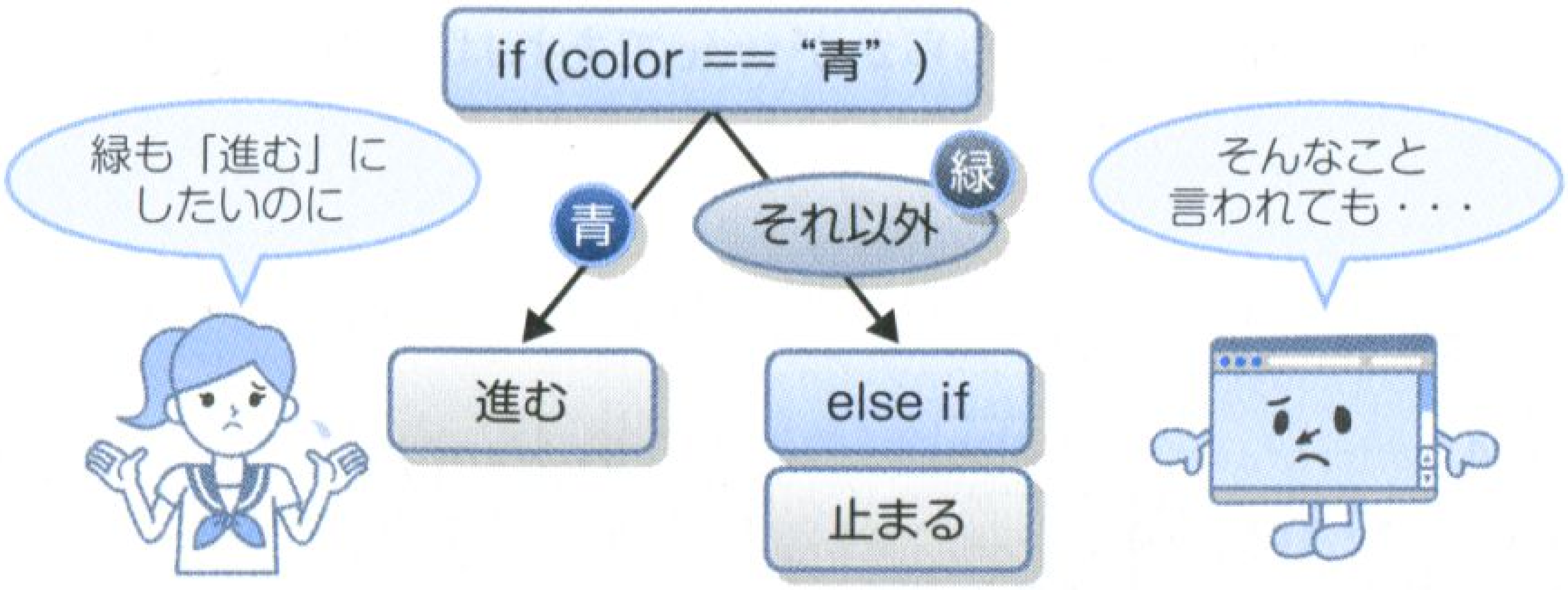
```
if(x == 10)  // 変数xの中身と10を「比較」
if(x = 10)   // 10を変数xに「代入」
```

Note

以降のサンプルコードでは、JavaScript（リスト5.1の網掛け部分）のみを掲載します。

より複雑な条件式の書き方

リスト5.1は、信号が青であれば進み、そうでなければ止まるというスクリプトでした。しかし、信号機の直進可能を表わす色は「青」と表現する人もいれば「緑」と表現する人もいます。



「青」と「緑」の両方を「進む」にしたい

どちらも「進む」という同じ処理を行なうにもかかわらず、条件が1つでは表わしきれない場合、**論理演算子**を使って複数の条件を記述できます。

論理演算子

演算子	説明	使用例	結果
&&	左辺と右辺のどちらも true の場合は true	`1 == 1 && 2 == 2`	`true`
||	左辺と右辺のどちらかが true の場合は true	`1 == 2 || 2 == 2`	`true`
!	真偽値を反転させる	`!a`	a が false の場合は true、a が true の場合は false

リスト5.2は、変数colorに代入されている値が"青"もしくは"緑"であれば以降の処理を実行する、という意味になるので、青と緑のどちらであっても同じ結果を得ることができます。

リスト5.2　複雑な条件式

```
var color = "青";

if (color == "青" || color == "緑") {
    console.log("進む");
} else {
    console.log("止まる");
}
```

■実行結果

```
進む
```

||は、左右に記述した条件式のどちらかがtrueであれば、if文のブロックを実行します。

```
var color = "青";

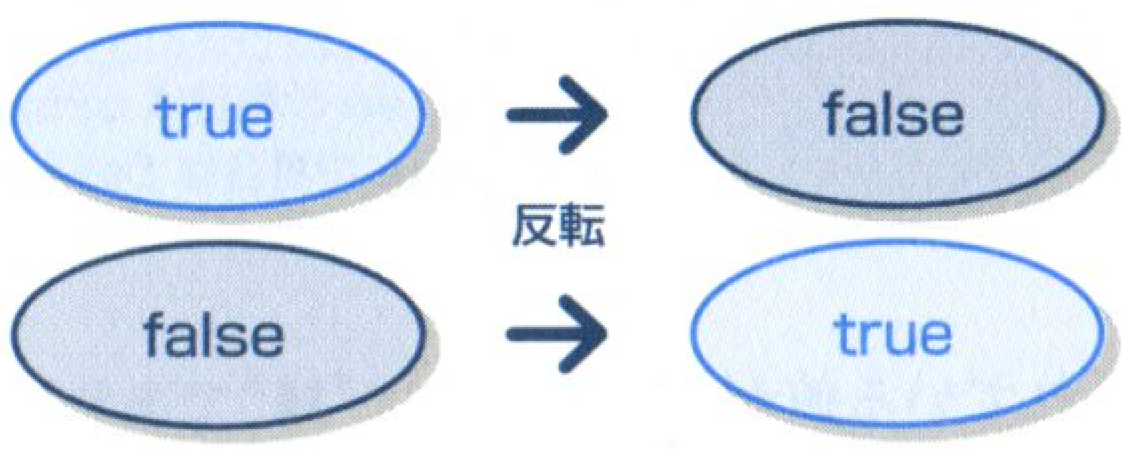
if (color == "青" || color == "緑") {
    console.log("進む");
} else {
    console.log("止まる");
}
```

true　false

どちらか片方でもtrueであれば
すぐ下のブロックを実行

◆ !演算子

論理演算子の中でも!演算子は少し特殊で、論理値を反転させる働きをします。



論理値を反転

JavaScriptでは、nullやundefinedなど値がないことを表わす特殊なものはfalseと等しい値として扱われます（データ型は異なります）。この特徴を活かし、変数の中に値が代入されているかどうか、という判断を行なう場合に、!演算子を使う方法がよく用いられます。

リスト5.3　!演算子の利用

```
var a;

if (!a) {
    console.log("aは空です");
}
```

■実行結果

```
aは空です
```

リスト5.3では、空の変数aの状態を反転させています。宣言だけして初期化されていない変数には**undefined**という特殊な値が既定値（初期値）として入ります。undefinedはfalseと同じ扱いなので、これを反転させるとtrueとなり、if文のブロック内が実行されます。

falseとして解釈される値には、他にも数値の0や空文字（文字列の長さが0文字）などがあります。

```
f (!a) {
  ↓ ❶aの中身はundefined
if (!undefined) {
  ↓ ❷undefinedはfalseと同等
if (!false) {
  ↓ ❸falseを反転
if (true) {
```

リスト5.3の変数aの反転動作イメージ

多方向分岐

ここまでに実行したスクリプトは、信号の状態に応じて「進む」か「止まる」のどちらかを行なう、2方向へ分岐する処理でした。ここにたとえば信号機の色が黄色だったら急いで進む、という処理を追加して3方向以上へ分岐させたい場合は、if文に**else if**（エルス イフ）ブロックを付けます。

文法 else ifブロックを利用した多方向分岐

```
if (条件式1) {
    条件式1の条件を満たす場合に実行する処理
} else if (条件式2) {
    条件式2の条件を満たす場合に実行する処理
} else if (条件式3) {
    条件式3の条件を満たす場合に実行する処理
} else {
    すべての条件式の条件を満たさない場合に実行する処理
}
```

else ifブロックは、複数記述できます。条件式1～3のうち複数の条件に一致する場合は、一番上にあるブロックのみ実行されます。

リスト5.4　if文による多方向分岐

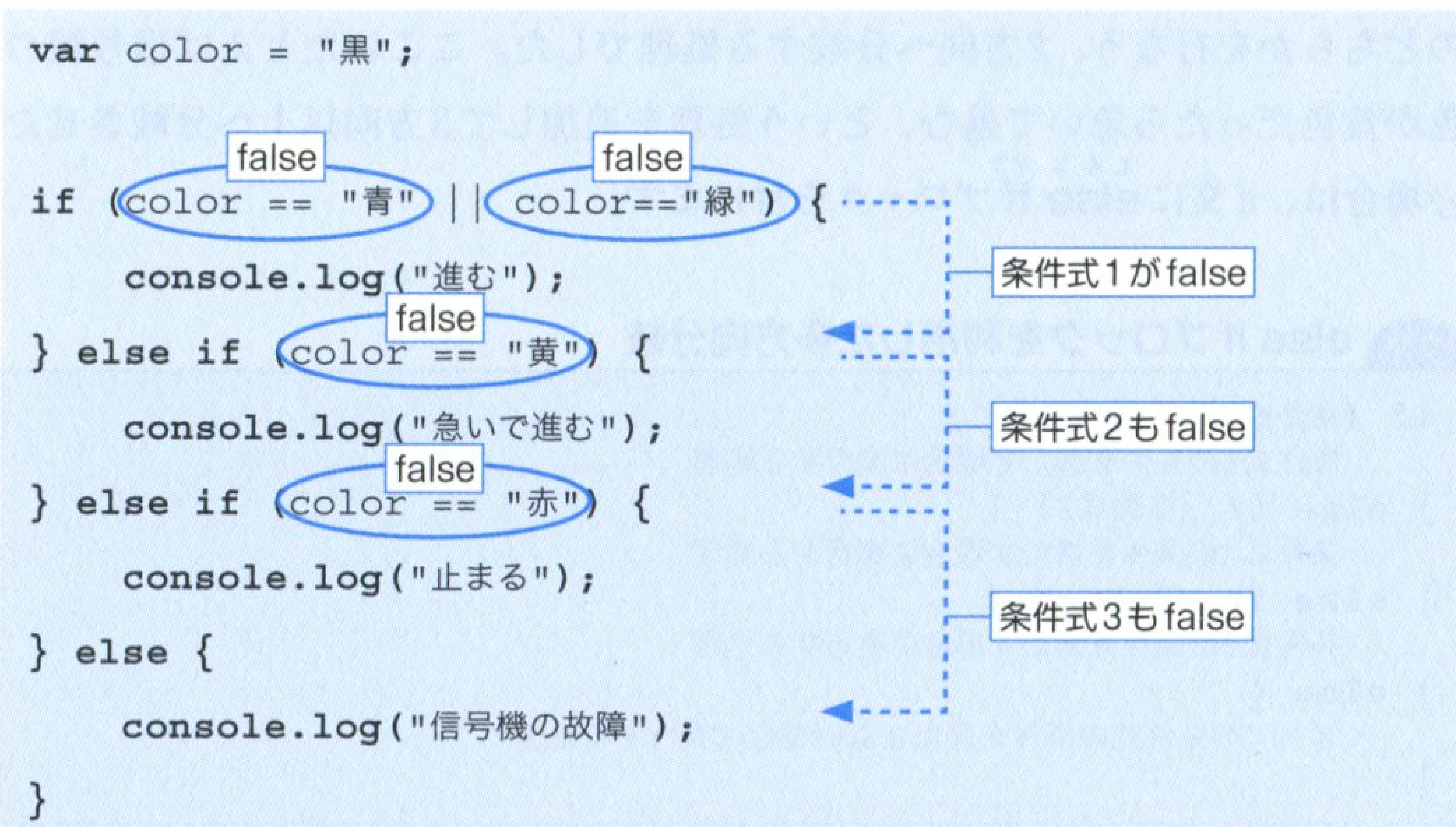
```
var color = "黒";

if (color == "青" || color=="緑") {
    console.log("進む");
} else if (color == "黄") {
    console.log("急いで進む");
} else if (color == "赤") {
    console.log("止まる");
} else {
    console.log("信号機の故障");
}
```

■実行結果

```
信号機の故障
```

リスト5.4は、信号機の色が青または緑の場合、黄色の場合、赤の場合、その他の色の場合でそれぞれ異なる処理を行なっています。この例では、変数colorの中身は黒なのでどの条件式にも当てはまらず、最後のelseブロック内が実行されます。



Point!



- 「分岐」とは、処理が多方向に分かれること。
- if文でスクリプトの分岐構造を作ることができる。
- 条件式は比較演算子を使って記述され、結果が論理値となる。

■ Check Test 1

Q1 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
var x = "ABC";
if (x != "ABC") {
    console.log("OK");
} else {
    console.log("NG");
}
```

Q2 以下のスクリプトを実行すると、どのような結果が表示されますか？

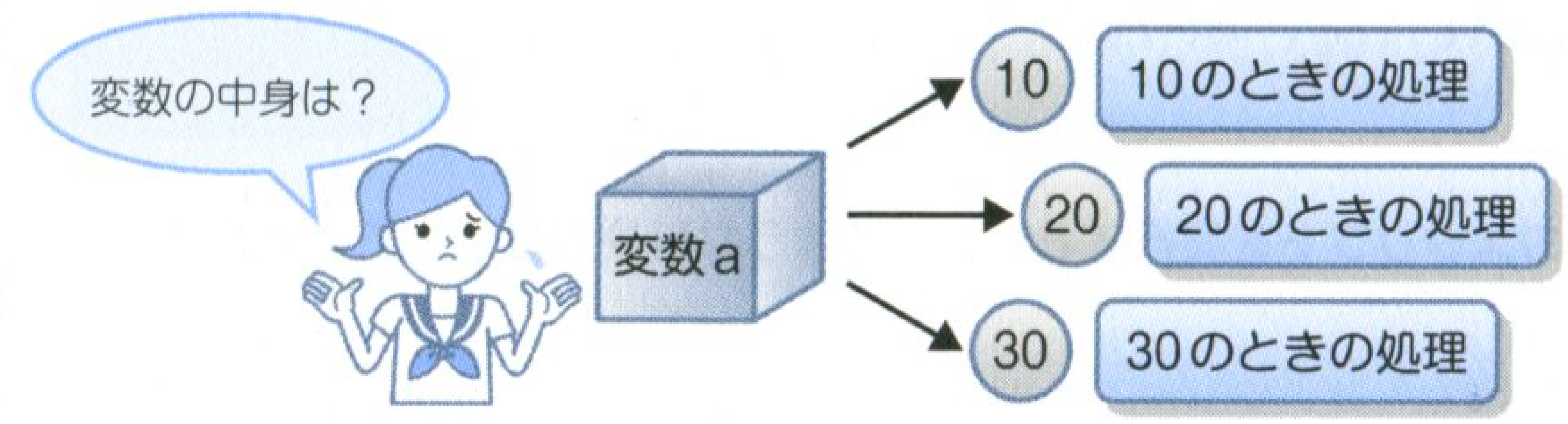
```
var x = 10;
var y = 20;
if (y >= 10 && x > y) {
    console.log("OK");
} else {
    console.log("NG");
}
```

➲解答は巻末に掲載

5-2 switch

switch文

JavaScriptで分岐構造を実現する、もう1つの構文を見ていきましょう。**switch**（スイッチ）**文**は、if文と同様に分岐処理を行ないます。if文をより簡略化してすっきりと記述するための構文です。ある変数の中身がどの値に一致するか、という判断を行ないます。



switch文は変数の値で分岐する

文法 switch文

```
switch (変数) {
    case 値1:
        変数の中身が値1に一致した場合の処理
    case 値2:
        変数の中身が値2に一致した場合の処理
    case 値3:
        変数の中身が値3に一致した場合の処理
    default:
        どの値にも一致しなかった場合の処理
}
```

switch文は、いずれかの値に一致すると、以降の処理がすべて実行されるという特徴があります。リスト5.5は、変数aに2が代入されているので、「case 2:」以降の処理がすべて実行されます。

リスト5.5 switch文の特徴

```
var a = 2;
switch(a) {
    case 1:
        console.log("aは1です");
    case 2:
        console.log("aは2です");
    case 3:
        console.log("aは3です");
    default:
        console.log("一致なし");
}
```

一致 → case 2: 以降すべて実行

■実行結果

```
aは2です
aは3です
一致なし
```

◆ break文

リスト5.5の実行結果に「aは2です」のみが表示されるようにする場合は、**break文**（ブレイク）を利用します。break文は、ブロックから強制的に抜けるときに使う構文です。

文法 break文

```
break;
```

リスト5.6 break文の利用

```
var a = 2;

switch(a) {
    case 1:
        console.log("aは1です");
        break;
    case 2:
        console.log("aは2です");
        break;
    case 3:
        console.log("aは3です");
        break;
    default:
        console.log("一致なし");
}
```

一致 → case 2: 〜 break; switch文のブロックから抜ける

■実行結果

```
aは2です
```

また、break文を利用すると、前節のリスト5.4のコードを、if文からswitch文に書き換えることができます。

リスト5.7 switch文による多方向分岐

```
var color = "黒";

switch(color) {
    case "青":
    case "緑":
        console.log("進む");
        break;
    case "黄":
        console.log("急いで進む");
        break;
    case "赤":
        console.log("止まる");
        break;
    default:
        console.log("信号機の故障");
}
```

■実行結果

```
信号機の故障
```

switch文はあくまでif文をシンプルに表現するための構文なので、switch文で表現できることはif文でも表現できます。逆に、if文で表現できることがswitch文でも表現できるとは限りません。switch文では、値の大小やデータ型の比較ができないからです。よって、分岐処理はほとんどの場合にif文で記述しますが、単純な条件式による分岐の場合はswitch文で簡潔に記述しても良いでしょう。

Point!



- 分岐処理を行なう方法には、if文とswitch文がある。
- switch文は単純な分岐を簡潔に記述したいときに使おう。

■ Check Test 2

Q1 以下のswtich文を実行した結果、表示される値はなんですか？

```
var x = "C";
switch(x) {
    case "A":
        console.log("A");
    case "B":
        console.log("B");
    case "C":
        console.log("C");
    case "D":
        console.log("D");
    default:
        console.log("E");
}
```

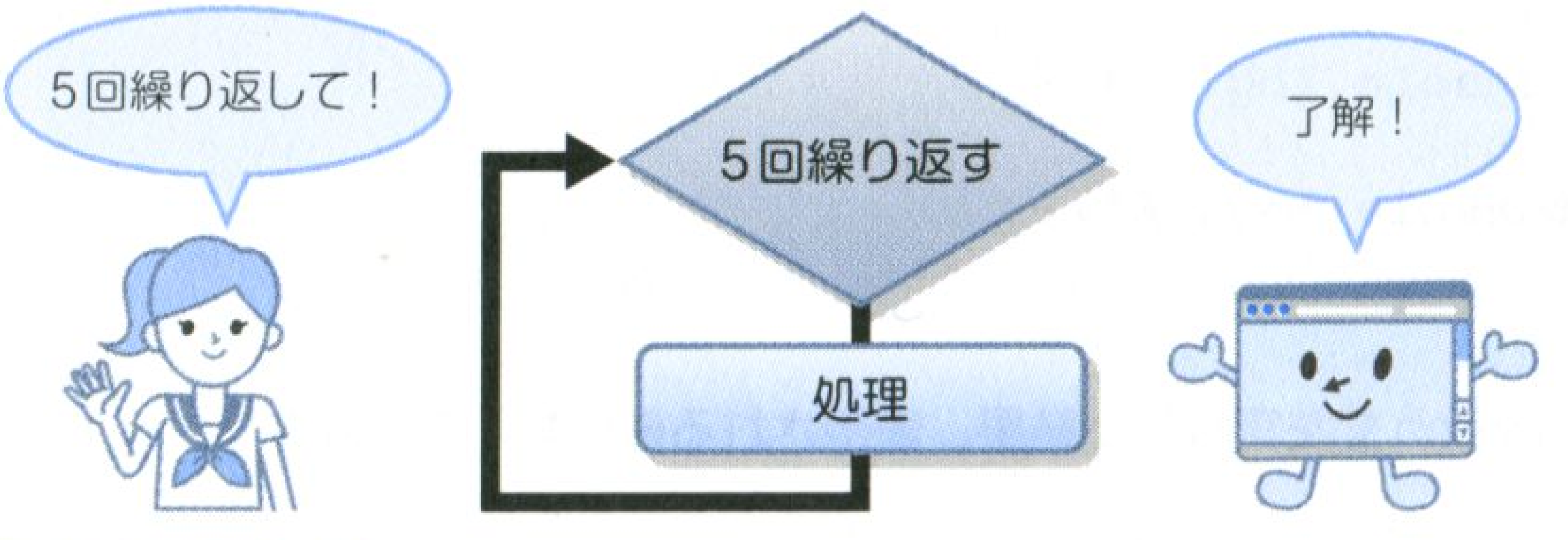
Q2 以下のswtich文を実行した結果、表示される値はなんですか？

```
var x = "Z";
switch(x) {
    case "A":
        console.log("A");
    case "B":
        console.log("B");
    case "C":
        console.log("C");
    case "D":
        console.log("D");
    default:
        console.log("E");
}
```

5-3 while文

反復処理とは

ある処理を5回実行したい、といった場合にコピー＆ペーストして5行分同じ記述をするのはナンセンスです。後から修正をすることになった場合、5行すべてを直さなくてはなりません。そこで、**反復**（はんぷく）あるいは**繰り返し**と呼ばれる構造を利用します。JavaScriptで反復処理を行なう方法には、**while文**（ホワイル）と**for文**（フォー）の2つがあります。



反復処理のイメージ

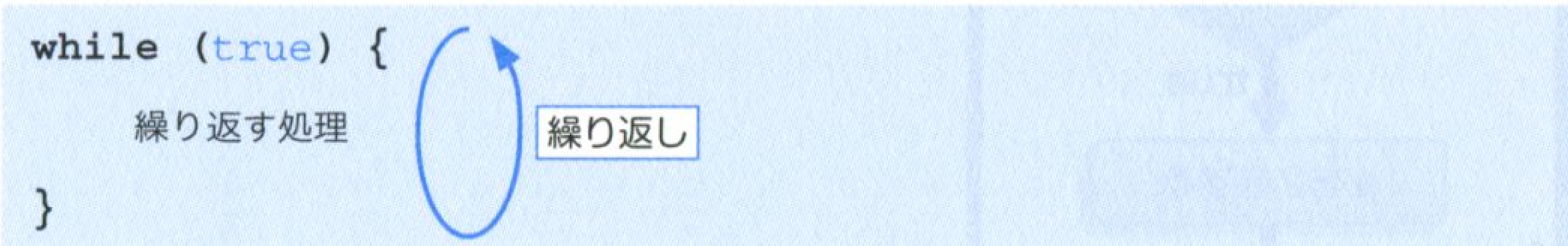
> **Note** while文やfor文のように同じことを繰り返すことをループとも呼びます。

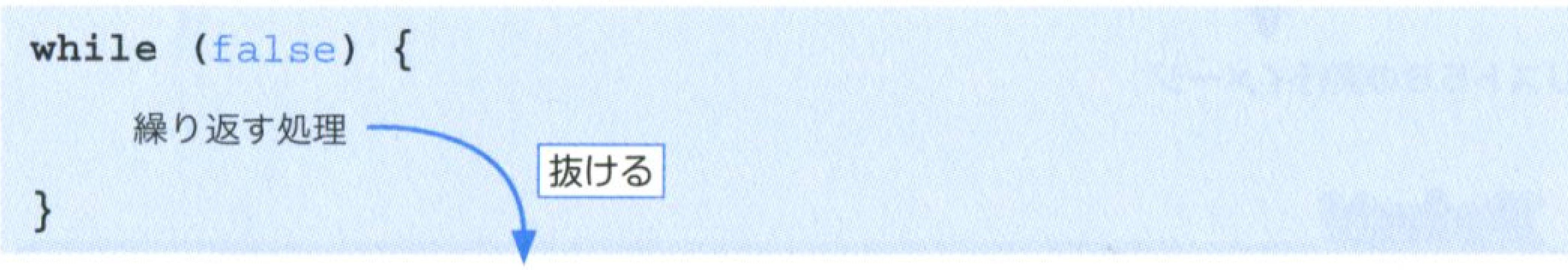
まずは、while文の使い方をおぼえましょう。

文法 while文

```
while (継続条件式) {
    繰り返す処理
}
```

継続条件式には、if文と同様に結果が論理値となる式を記述します。継続条件式の結果がtrueの間ブロック内の処理を繰り返し、falseになると繰り返しを終了します。継続条件式は1回分の繰り返し処理の最初のタイミングで検証されます。





では、基本的なwhile文の例を見てみましょう。

リスト5.8　while文の利用

```
var a = 1;

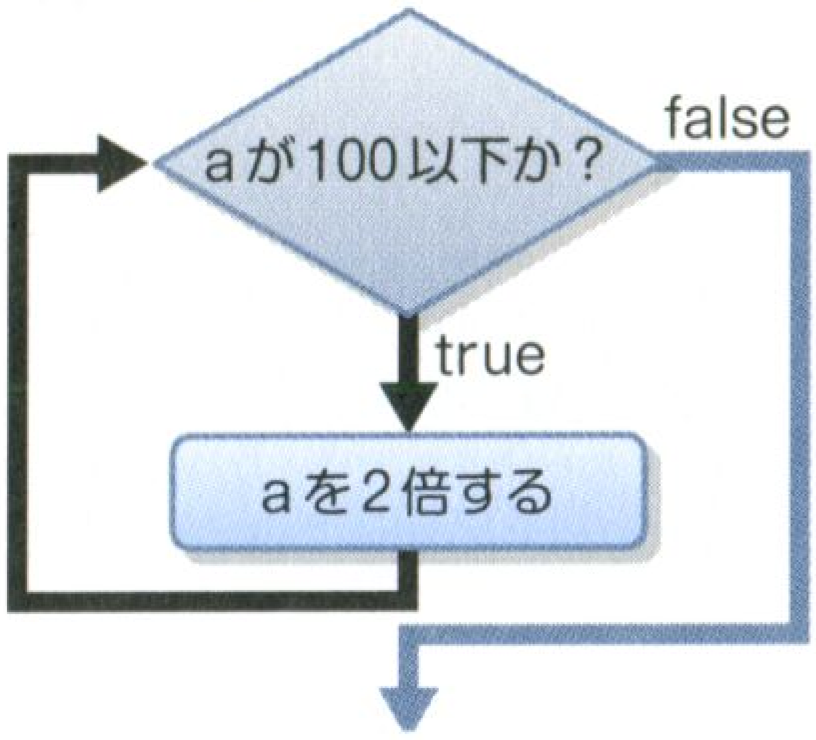
while (a <= 100) {
    a *= 2;
    console.log(a);
}
console.log("終了");
```

■実行結果

```
2
4
8
16
32
64
128
終了
```

リスト5.8は、変数aの値をどんどん倍にしていき、100を超えたら繰り返しを終了する、というスクリプトです。aの値が100を超えると、次の繰り返し処

理へ進んだタイミングで継続条件式の結果がfalseと判定され、繰り返しを抜けます。



リスト5.8の実行イメージ

Point!

- 反復（繰り返し）とは同じ処理を何度も繰り返すスクリプト構造のことで、ループとも呼ぶ。
- 反復処理を行なう方法にはwhile文とfor文がある。



Check Test 3

Q1 以下のwhile文を実行すると、繰り返しは何回行なわれますか？

```
var a = 0;
while(a < 3) {
    a++;
}
```

Q2 以下のwhile文を実行すると、繰り返しは何回行なわれますか？

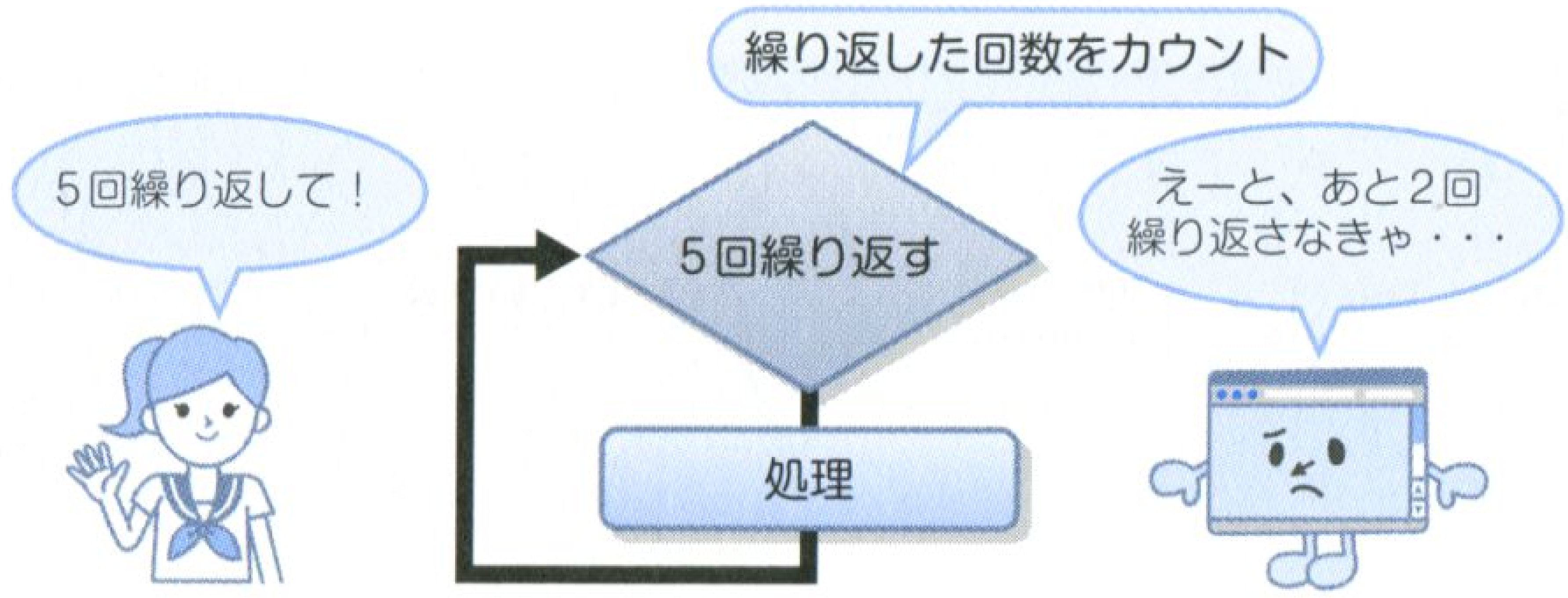
```
var a = 3;
while(a < 3) {
    a++;
}
```

解答は巻末に掲載

5 4 for文

for文とは

while文は、繰り返しの継続条件のみを指定する単純な構文でした。それに対して**for文**は少し複雑で、繰り返すたびに変数の値を増減させることができます。この変数を**カウンタ変数**と呼びます。何回目の繰り返し処理を実行しているのかという情報が必要なときなどにはfor文を利用すると良いでしょう。



for文のイメージ

for文は次のように記述します。

文法 for文

```
for (初期化式 ; 継続条件式 ; 増減式) {
    繰り返す処理
}
```

式の種類	説明	実行タイミング
初期化式	カウンタ変数を初期化する代入式	最初の1回だけ実行される
継続条件式	繰り返しを継続する条件式	1回分の繰り返し処理の最初に毎回実行される
増減式	カウンタ変数の値を増減させる計算式	1回分の繰り返し処理の最後に毎回実行される

それでは、基本的なfor文の例を見てみましょう。

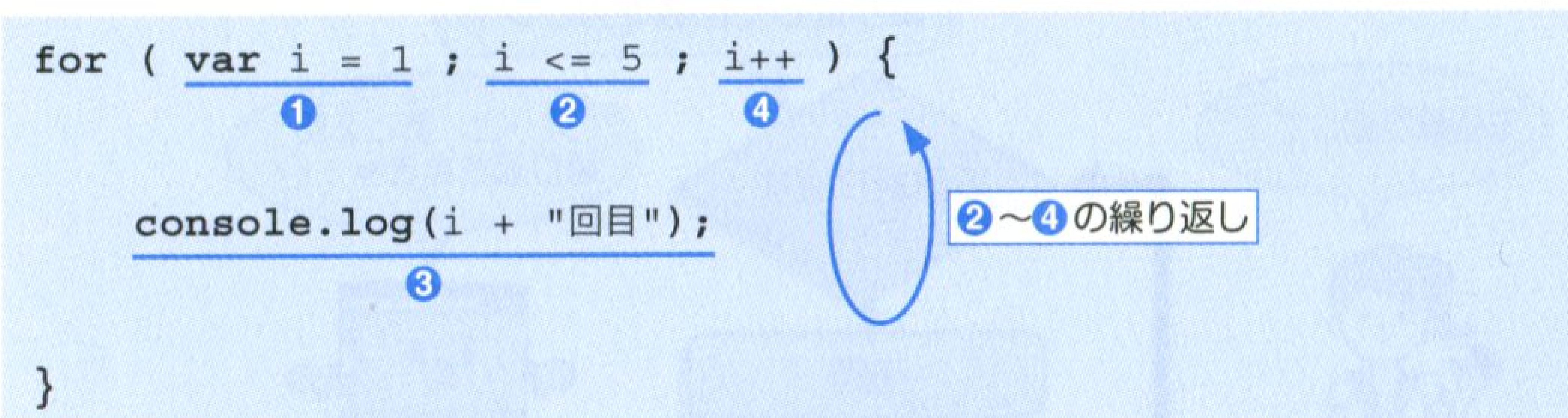
リスト5.9 for文の利用

```
for (var i = 1 ; i <= 5 ; i++) {
    console.log(i + "回目");
}
```

■実行結果

```
1回目
2回目
3回目
4回目
5回目
```


リスト5.9は次の順序で実行されます。



❶変数iに1が代入される（初期化式）

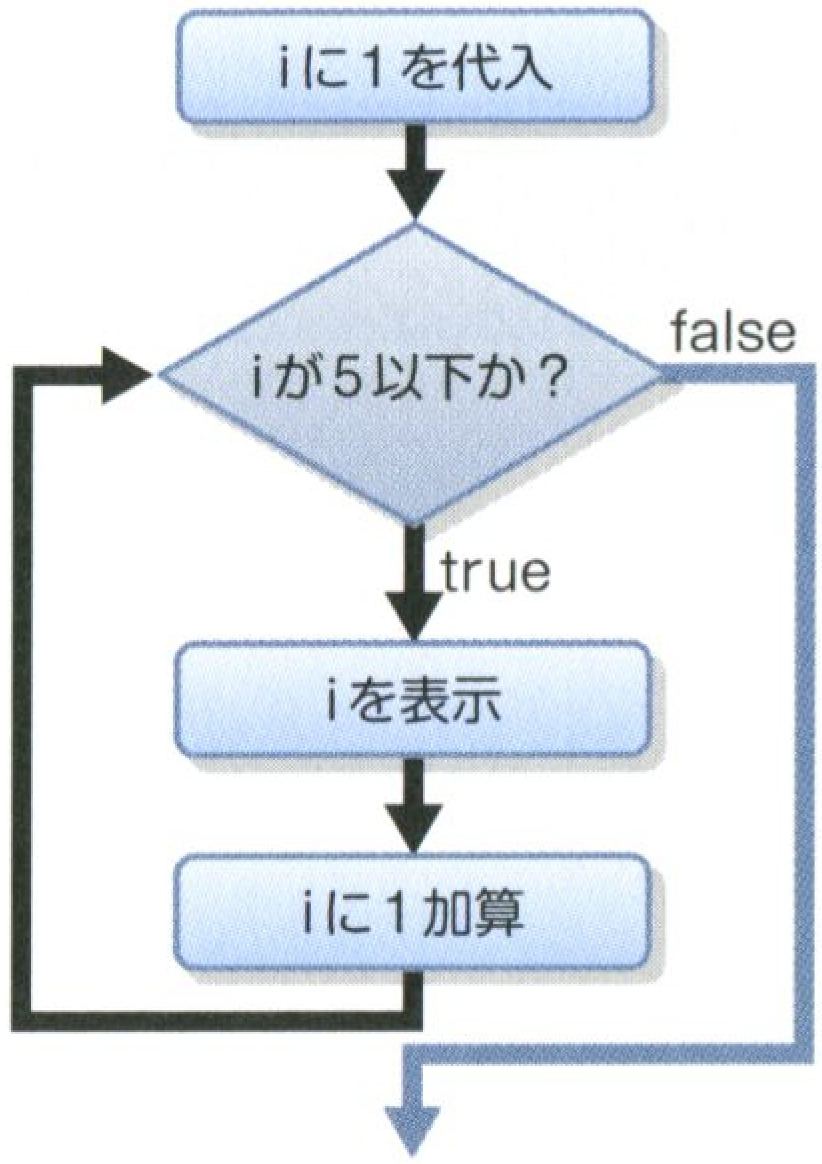
❷iが5以下かどうか検証される（継続条件式）

❸iの値がコンソールに表示される

❹iに1加算される（増減式）

❺継続条件式の結果がfalseになるまで❷〜❹を繰り返す

実行イメージは次のとおりです。



リスト5.9の実行イメージ

カウンタ変数の名前は通常の変数と同様に自由ですが、慣例としてiという名前がよく使われます。

for文の応用例

for文はさまざまな応用が可能で、3つの式の指定を工夫すれば、カウントアップではなくカウントダウンすることなども可能です。

■カウントダウン

```
for (var i = 5 ; i >= 1 ; i--) { }
```

また、for文は配列を扱う際にも重宝します。配列のインデックスをカウンタ変数で指定することで、配列の要素に対して1つずつ順に処理を実行できます。

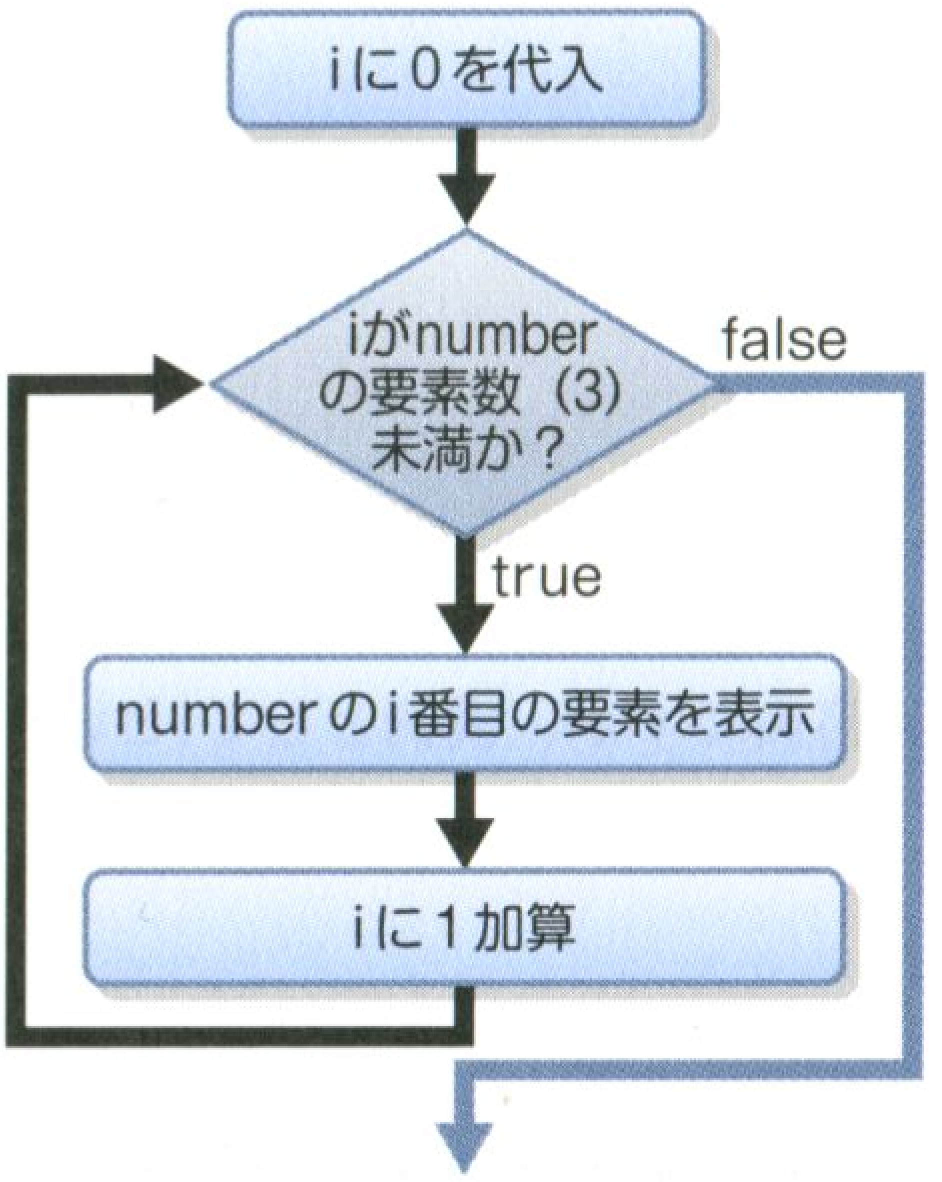
リスト5.10　配列とfor文

```
var number = [ 10, 20, 30 ];
for (var i = 0 ; i < number.length ; i++) {
    console.log(number[i]);
}
```

■実行結果

```
10
20
30
```

実行イメージは次のとおりです。

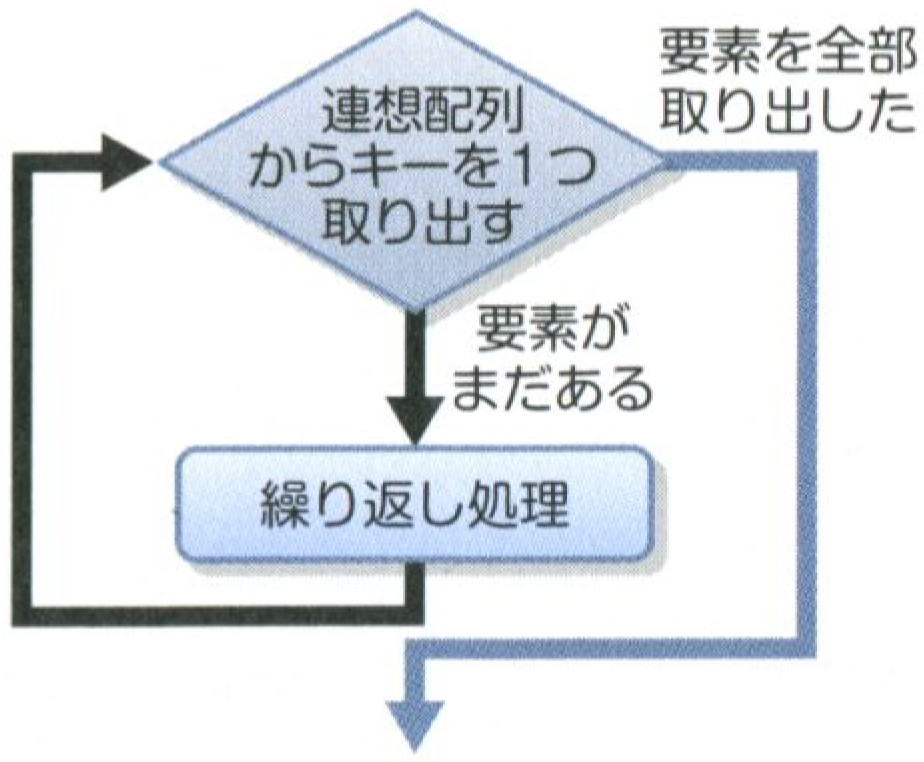


リスト5.10の実行イメージ

リスト5.10のnumber.lengthは、配列numberの要素数を表わします。よって、for文の継続条件は「iの値が0から3未満まで」になります。配列numberのインデックスは先頭から順に0、1、2ですから、ちょうどカウンタ変数の値がインデックスと対応するわけです。

for-in文

任意の文字列をキーとする連想配列は、通常のfor文では扱うことができません。連想配列の要素を1つずつ処理する場合、for文ではなく**for-in文**（フォー・イン）を利用します。



for-in文の実行順序

for-in文は次のように記述します。

文法 for-in文

```
for (var 変数名 in 連想配列名) {
    連想配列名[変数名]とすることで各要素を参照できる
}
```

for-in文は、連想配列の要素数分繰り返しを実行します。繰り返すたびに変数の中に連想配列のキーのうちいずれか1つが代入されます。

リスト5.11 連想配列とfor-in文

```
var favorites = {
    food : "カレーライス",
    color : "青",
    number : 7
};
for (var key in favorites) {
    console.log(key + ' : ' + favorites[key]);
}
```

■実行結果

```
food : カレーライス
color : 青
number : 7
```

実行イメージを見てみましょう。

```
var favorites = {
    food: "カレーライス",
    color: "青",
    number: 7
};
for (var key in favorites) {
    console.log(key + ' : ' + favorites[key]);
}
```

各キーが1回ずつ取り出され、keyに代入される

リスト5.11は、連想配列のすべての要素を一覧表示するスクリプトです。連想配列favoritesの要素数が3つなので、for-in文の繰り返し回数は3回です。1回目の繰り返し処理では、変数keyの中に「food」が代入されます。2回目は「color」、3回目は「number」というように、1回分の繰り返しごとにキーが1つずつ取得されます。

なお、キーが取得される順番は保障されていないため、配列の先頭要素から順序通りに処理をしたい場合にはfor-in文は適しません。

Point!

- while文に「カウント機能」を付け加えたものがfor文である。
- 配列の要素を1つずつ処理する場合はfor文を使う。
- 連想配列の要素を1つずつ処理する場合はfor-in文を使う。



Check Test 4

Q1 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
for (var i = 5 ; i > 0 ; i--) {
    console.log(i);
}
```

Q2 以下のfor-in文を実行すると、繰り返しは何回行なわれますか？

```
var number = { a : 100, b : 200 , c : 300, d : 400 };
for (var key in number) {
    console.log(number[key]);
}
```

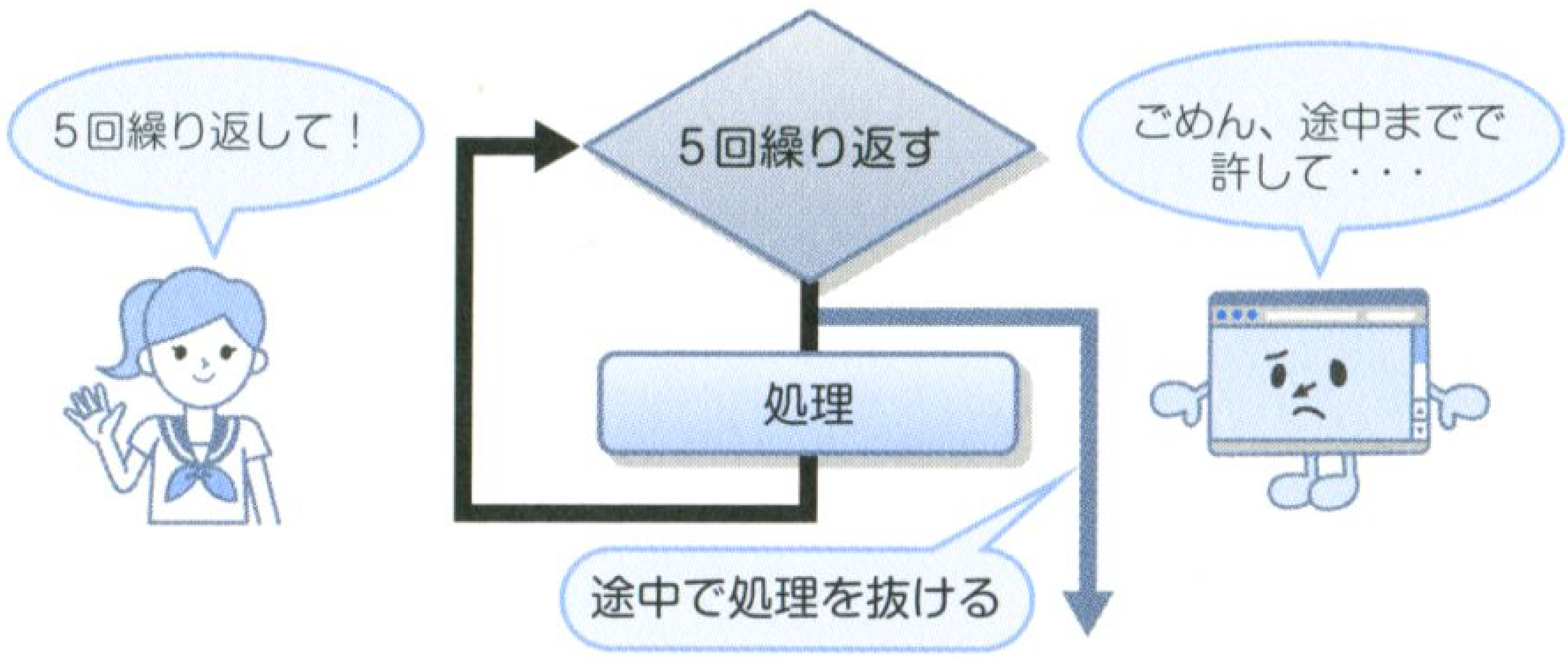
➲解答は巻末に掲載

5 繰り返しの制御

繰り返しの途中で処理を中断したり強制終了したりするための**break文**と**continue**（コンティニュー）**文**を学びます。

繰り返しの中のbreak文

switch文の節で学習したbreak文は、ブロックの外へ抜けるための構文でした。このbreak文をwhile文やfor文のブロック内に記述することで、継続条件式がtrueであっても強制的にブロックの外へ抜けることができます。繰り返しの途中で、例外的に繰り返しを終了させたい場合などに使います。



break文のイメージ

for文での利用例を見てみましょう。

リスト5.12 for文とbreak文

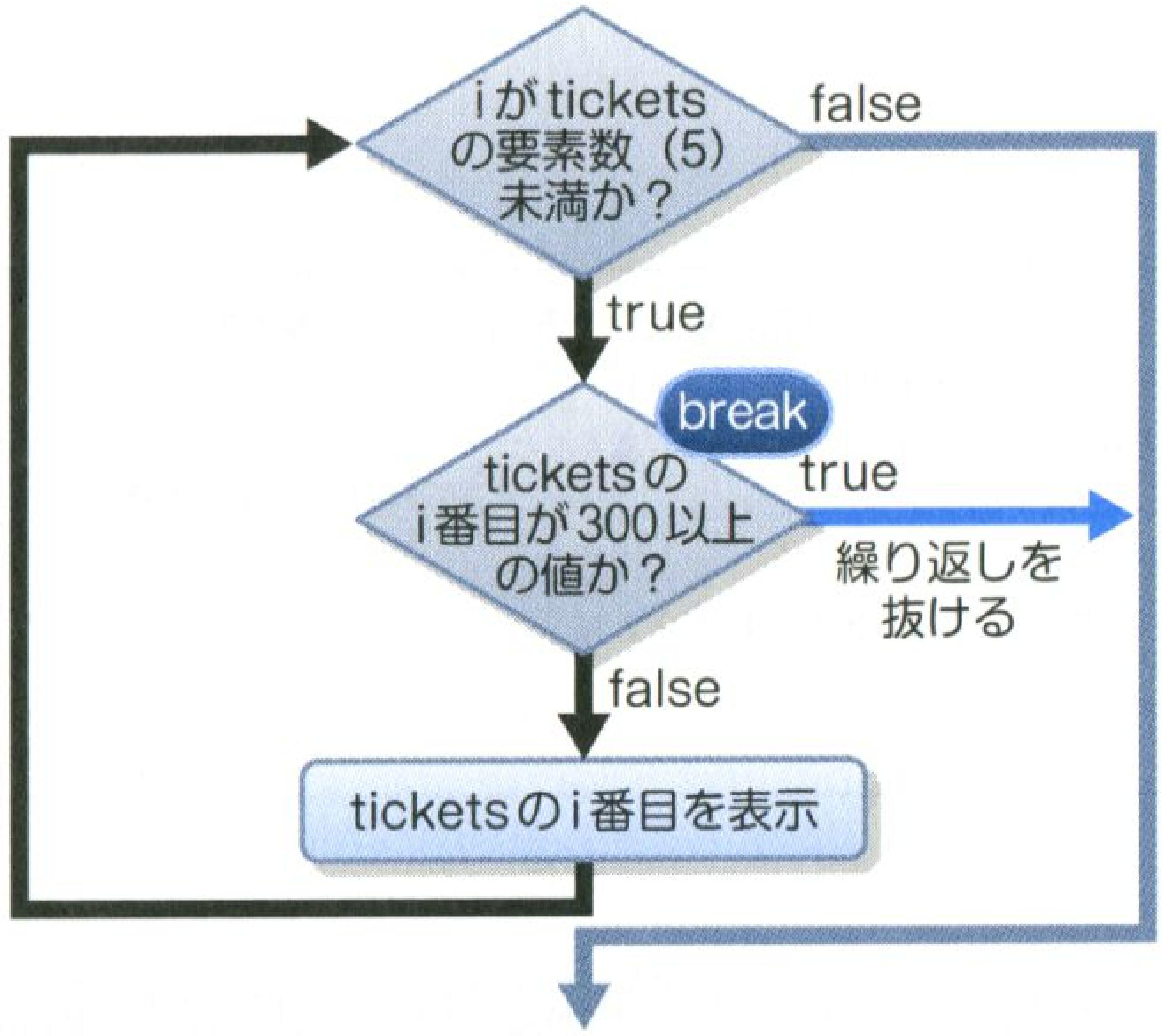
```
var tickets = [ 290, 296, 299, 301, 305 ];
for (var i = 0; i < tickets.length; i++) {
    if (tickets[i] >= 300) {
        break;
    }
    console.log(tickets[i]);
}
console.log("終了");
```

繰り返しを抜ける

■実行結果

```
290
296
299
終了
```

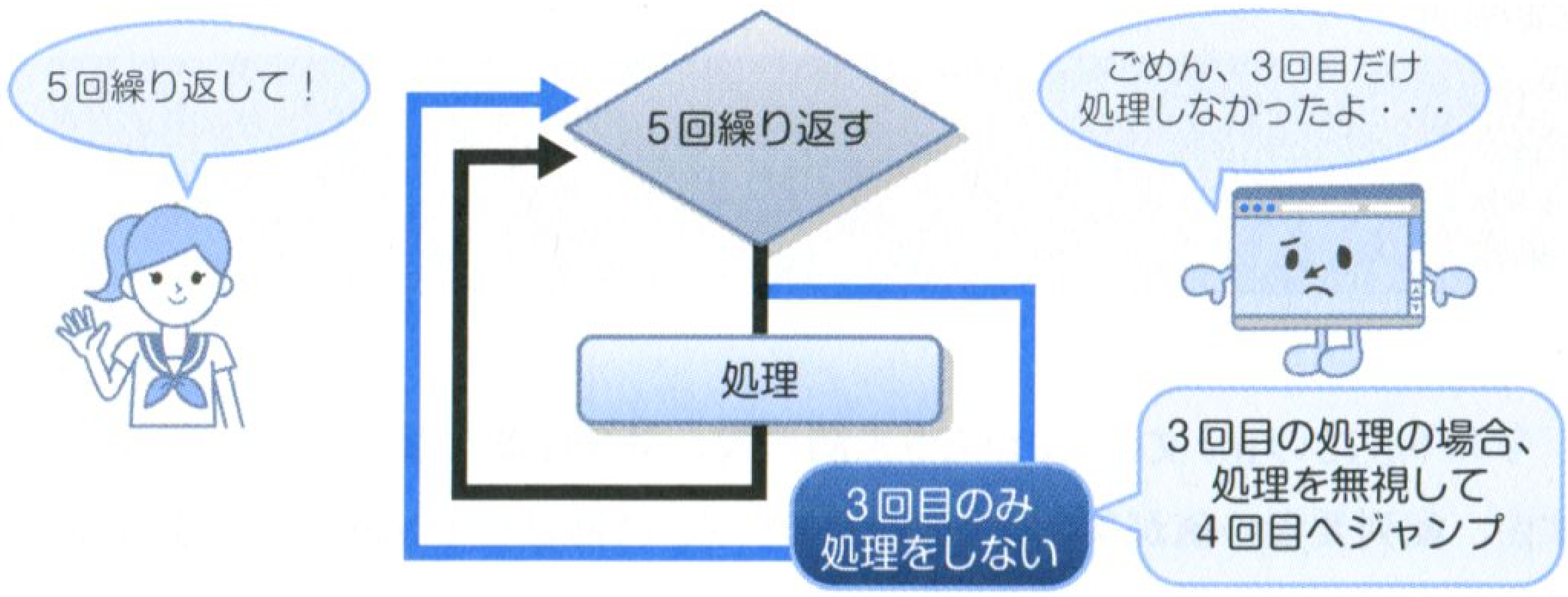
リスト5.12のfor文は、0から5未満の間、計5回繰り返す条件になっていますが、配列要素の値が300以上になった時点（4回目）でbreak文によって繰り返しを抜けています。



リスト5.12の実行イメージ

continue文

continue文は、1回分の繰り返し処理を中断し、次の繰り返し処理へ進む構文です。break文とは違って継続条件式がfalseになるまで繰り返しを続けます。最後まで繰り返すけれど、ある条件のときにだけ処理を行ないたくない場合などに利用します。



continue文のイメージ

ではfor文での例を見てみましょう。

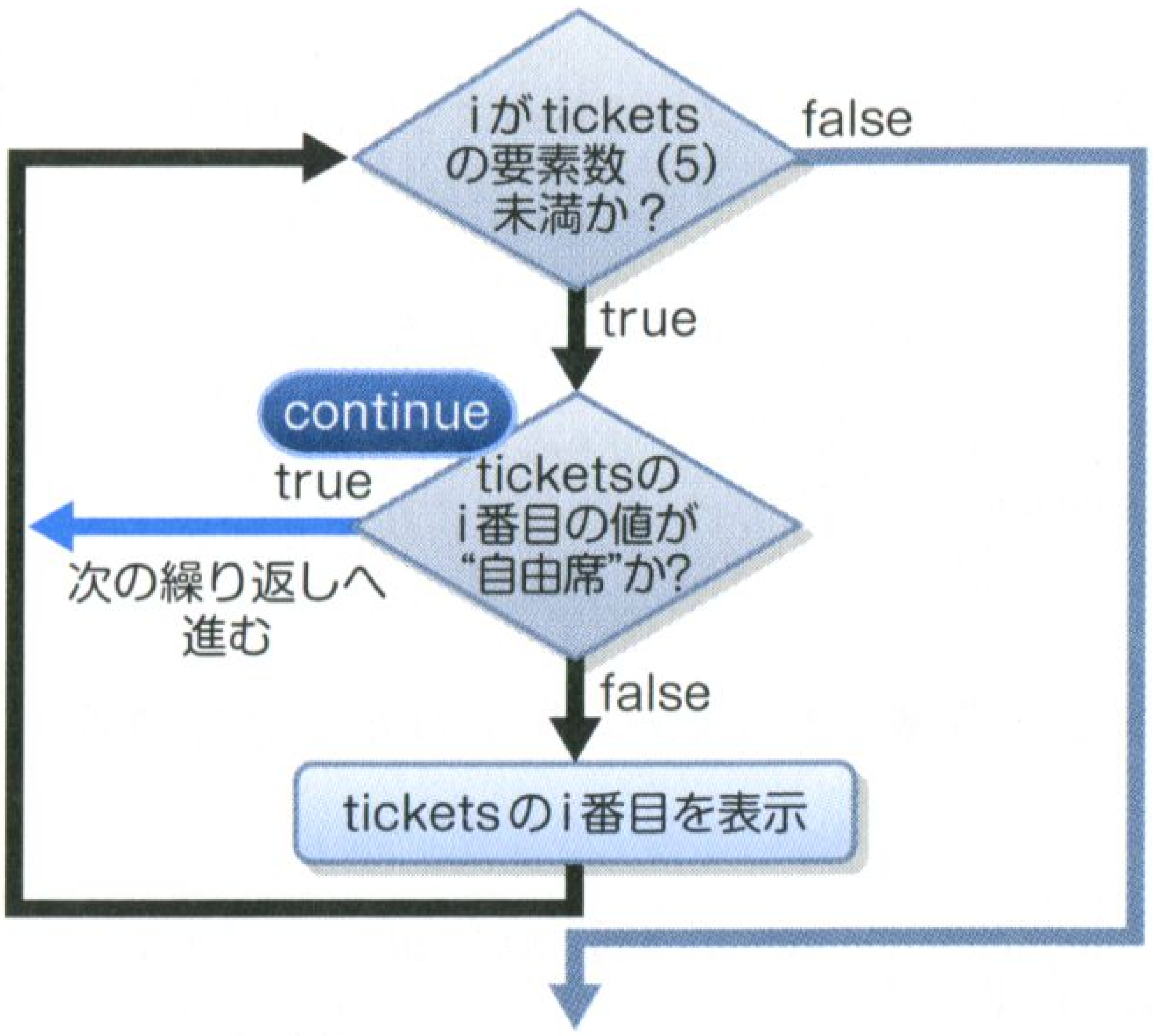
リスト5.13　for文とcontinue文

```
var tickets = [ "指定席", "自由席", "指定席", "自由席", "指定席" ];
for (var i = 0; i < tickets.length; i++) {
    if (tickets[i] == "自由席") {
        continue;                          次の繰り返し処理へ
    }
    console.log(i + " : " + tickets[i]);
}
console.log("終了");
```

■実行結果

```
0 : 指定席
2 : 指定席
4 : 指定席
終了
```

リスト5.13は、配列要素の値が"自由席"だった場合にのみcontinue文を実行するようになっています。continue文が実行されると、以降の処理は無視してただちに次の繰り返し処理へ進むため、コンソールへの出力は行なわれません。



リスト5.13の実行イメージ

Point!

- 繰り返しの途中で処理を抜けるにはbreak文を使う。
- 1回分の繰り返し処理をスキップするにはcontinue文を使う。



この章の到達度チェック

Q1 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
var x = 10;
if (x === "10") {
    console.log("OK");
} else {
    console.log("NG");
}
```

Q2 switch文において、if文のelseと同じ役割を果たすキーワードはなんですか？

Q3 配列fruitsの要素を一覧表示するスクリプトになるように、空欄AとBを埋めてください

```
var fruits = [ "りんご", "みかん", "パイナップル", ⏎
"ぶどう", "バナナ" ];
for (var i = [  A  ] ; i < [  B  ] ; i++) {
    console.log(fruits[i]);
}
```

Q4 連想配列userの要素の値を一覧表示するスクリプトになるように、空欄AとBを埋めてください。

```
var user = { name : "山田太郎", age : 25, height : 170 };
for(var x [  A  ] user) {
    console.log(user[  B  ]);
}
```

Q5 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
for (var i = 1; i <= 10; i++) {
    if (i > 5) {
        break;
    } else if (i % 3 == 0) {
        continue;
    }
    console.log(i);
}
```

Q6 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
for (var i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        break;
    }
    console.log(i);
}
```

Q7 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
for (var i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        continue;
    }
    console.log(i);
}
```

➲解答は巻末に掲載

第6章

関数

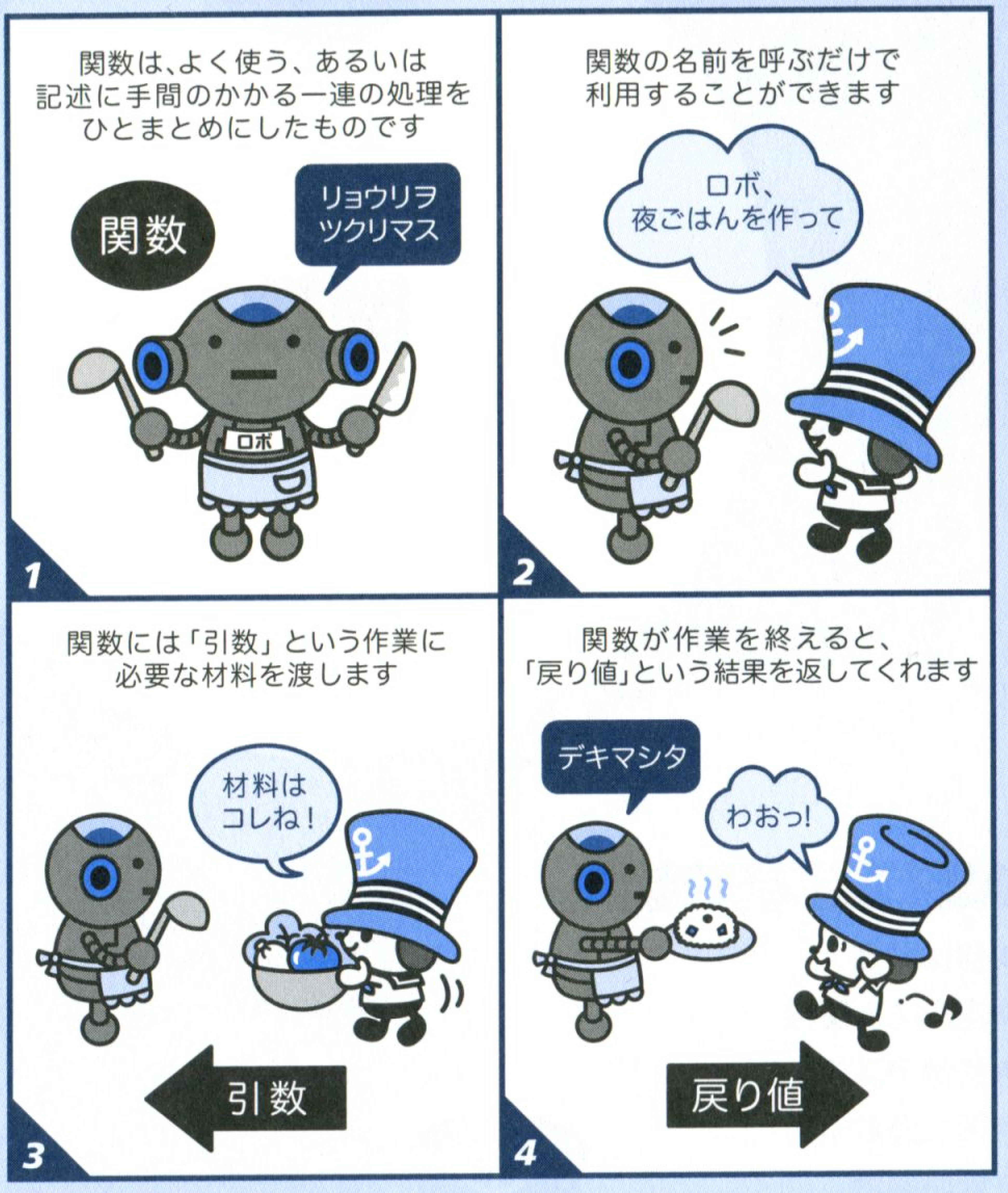
この章で学ぶこと

- 自作関数
- 組み込み関数
- 変数のスコープ
- 匿名関数
- 即時関数



6

関数は材料を渡すと処理をしてくれる便利なヤツ



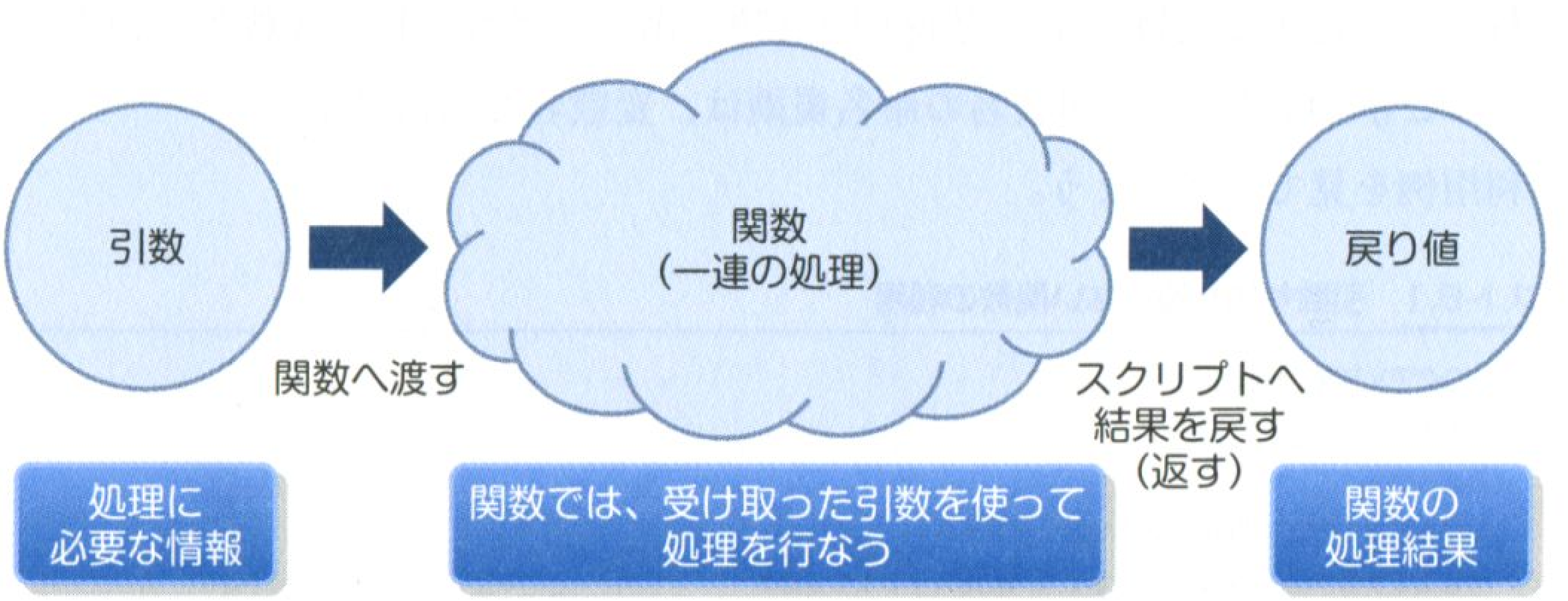
JavaScriptのスクリプトは、複数の関数の組み合わせによって構成されます。よく使う機能なので、本章で仕組みと記述方法をしっかりと理解しておきましょう。

6-1 関数～自作関数

関数とは

一連の処理をひとまとめにして、名前を付けたものを**関数**と呼びます。手間のかかる処理を関数として定義（作成）しておけば、必要なときに何度でも呼び出すことができます。同じ処理を何度も実行したり、一部条件を変えて実行したいときなどに有効です。また、スクリプトが長くなると可読性が低くなるため、機能や役割ごとに分けて関数化し、何の処理がどこに記述されているのかをわかりやすくする目的もあります。

関数は、処理の材料となる**引数**を受け取ります。そして、処理が終了すると**戻り値**と呼ばれる処理結果を返します。



関数の機能

関数には、大きく分けて次の2種類があります。

- **自作関数** ………自分で定義（作成）して使うもの
- **組み込み関数** …プログラミング言語にすでに組み込んであるもの（JavaScriptがあらかじめ備えている関数）

最初にこの節では、関数の知識の基本となる自作関数の作り方、すなわち関数の定義方法について学びます。

引数も戻り値もない関数の書き方

関数の引数と戻り値はいずれも省略することができます。まず、最も単純な、引数も戻り値もない関数の定義方法から見ていきましょう。

文法 関数の定義 引数・戻り値なし

```
function 関数名() {
    処理;
}
```

文法 定義した関数の呼び出し 引数・戻り値なし

```
関数名();
```

関数は、function（ファンクション）文を使って定義します。定義した関数を実行するときは、関数名を記述するだけです。定義の際も呼び出しの際も、必ず関数名の後ろに丸かっこ()を付けます。関数名の命名規則は、変数名と同様です。

利用例を見てみましょう。

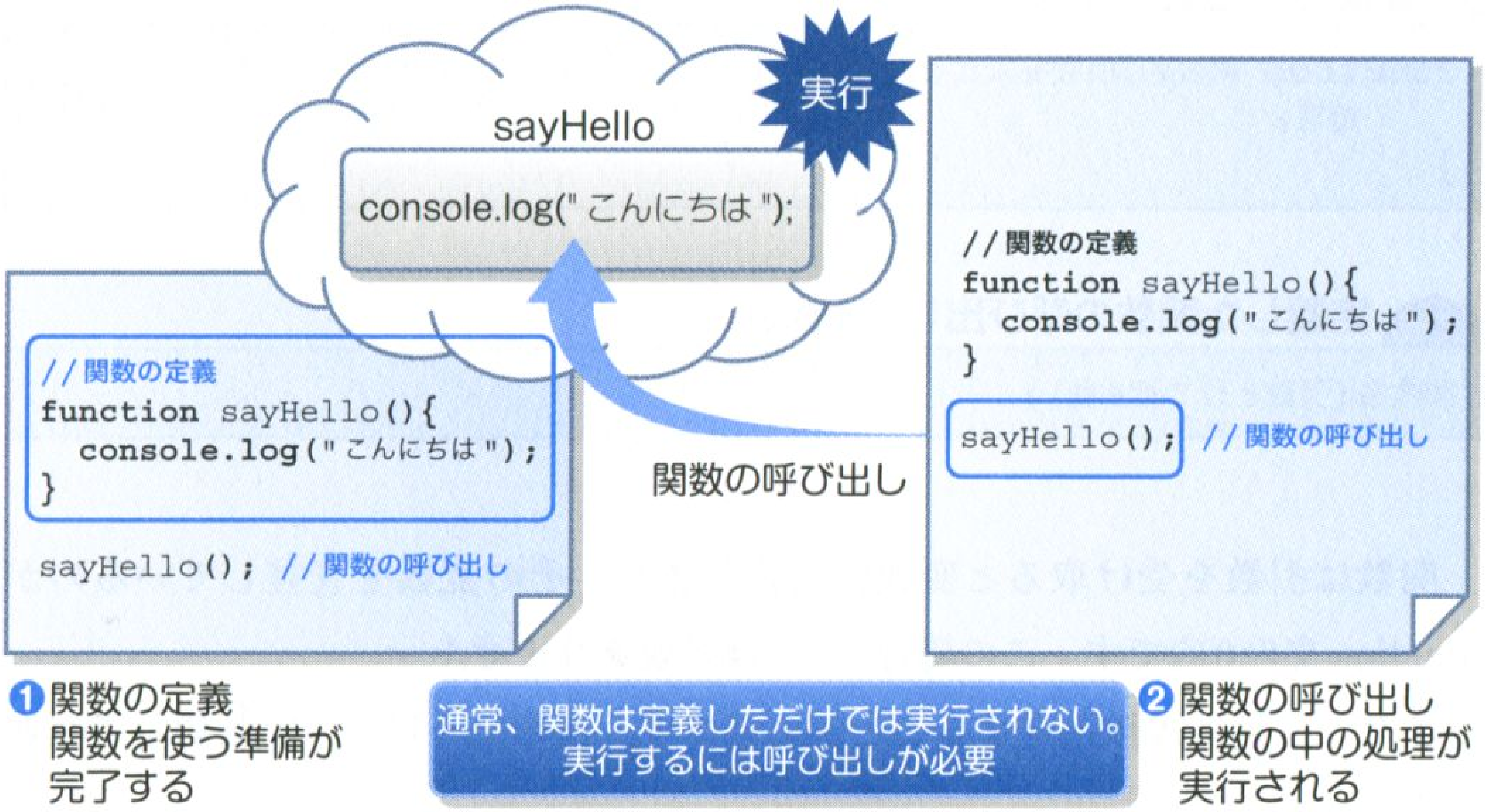
リスト6.1 引数も戻り値もない関数の利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   // 関数の定義
   function sayHello() {
       console.log("こんにちは");
   }
   sayHello();     // 関数の呼び出し
  </script>
 </head>
 <body>
 </body>
</html>
```

■実行結果

```
こんにちは
```

リスト6.1では、sayHelloという名前の関数を定義しています。関数は、定義しただけではまだ実行されません。関数を呼び出したタイミングで初めて実行されます。



関数の実行イメージ

なお、関数の定義は記述順序に関係なく優先的に行なわれるので、次のように関数の呼び出しが先に記述されていても問題なく動作します。

```
sayHello();   // 関数の呼び出し
// 関数の定義
function sayHello() {
    console.log("こんにちは");
}
```

Note 以降のサンプルコードでは、JavaScript（リスト6.1の網掛け部分）のみを掲載します。

引数のある関数の書き方

引数とは、関数に渡す材料のことです。たとえば、ある値を2倍にする処理を行なう関数があった場合、「ある値」とはいくつなのかを指定しなければなりません。この情報が引数です。引数は、複数指定することができます。

文法 **関数の定義** 引数あり

```
function 関数名(引数を入れる変数名) {
    処理;
}
```

文法 **定義した関数の呼び出し** 引数あり

```
関数名(引数として渡す値);
```

関数は引数を受け取ると変数に代入します。その変数を宣言しているのがfunction文の()内です。この場合、varは必要ありません。

また、引数は複数指定することが可能です。次のようにカンマ区切りで記述します。

文法 **関数の定義** 引数の複数指定

```
function 関数名(引数1, 引数2 … ) {
    処理;
}
```

文法 **定義した関数の呼び出し** 引数の複数指定

```
関数名(引数1, 引数2 … );
```

では、引数のある関数の例を見てみましょう。

リスト6.2 引数のある関数の利用

```
function sayHello(name) {
    console.log(name + "さん、こんにちは");
}

sayHello("よしお");
```

■実行結果

```
よしおさん、こんにちは
```

リスト6.2を実行すると、"よしお"という文字列が引数として関数に渡され、nameに代入されます。sayHello関数内では、nameは通常の変数と同様に扱うことができます。

```
function sayHello(name) {
    console.log(name + "さん、こんにちは");
}

sayHello("よしお");
```

渡される

戻り値のある関数の書き方

戻り値は、関数の処理結果です。たとえば、1～10までの数値の合計値を求める関数があった場合、算出した合計値を関数の呼び出し元に返却しなければなりません。この返却された値を戻り値と呼びます（返り値、返却値と呼ぶ場合もあります）。戻り値は引数と違って1つだけしか指定できません。

文法 関数の定義　戻り値あり

```
function 関数名() {
    処理;
    return 戻り値;
}
```

文法 定義した関数の呼び出し　戻り値あり

```
戻り値を入れる変数名 = 関数名();
```

6

return文が実行されると、関数の呼び出し元に戻ります。その際、戻り値として指定した値が返却されるので、戻り値を代入する変数を用意しておきましょう。

リスト6.3　戻り値のある関数の利用

```
function calc() {
    var num = 0;
    for(var i=1; i<=10; i++) {
        num += i;
    }
    return num;
}

var result = calc();
console.log(result);
```

■実行結果

```
55
```

リスト6.3のcalc関数は、1から10までの値を合算した結果を戻り値として返します。return文が実行されると、変数numの中身の55が呼び出し元に返却されます。次のように置き換わるイメージです。

■関数の実行前

```
var result = calc();
```

■関数の実行後

```
var result = 55;
```

このスクリプトの実行の流れを図示すると、次のようになります。

```
function calc() {
    var num = 0;
    for(var i=1; i<=10; i++) {
        num += i;
    }
    return num;
}
var result = calc();
console.log(result);
```

❶呼び出し
❷関数の実行
❸戻り値の返却

なお、戻り値を一度変数に代入しなくても、関数を呼び出して戻り値をコンソールに表示するだけであれば、次のように1行で記述することも可能です。

```
console.log(calc());
```

スクリプトは丸かっこ()の内側から実行されるので、まずcalc関数が実行され、次にコンソールへの出力が行なわれます。

```
console.log(calc());
```

↓ 内側のcalc()が実行され、戻り値が返却される

```
console.log(55);
```

実行イメージ

リスト6.4は、引数と戻り値の両方がある関数です。円の半径を引数として渡すと、円の面積を求めて戻り値として返します。このように、関数の中から別の関数を呼び出すことも可能です。

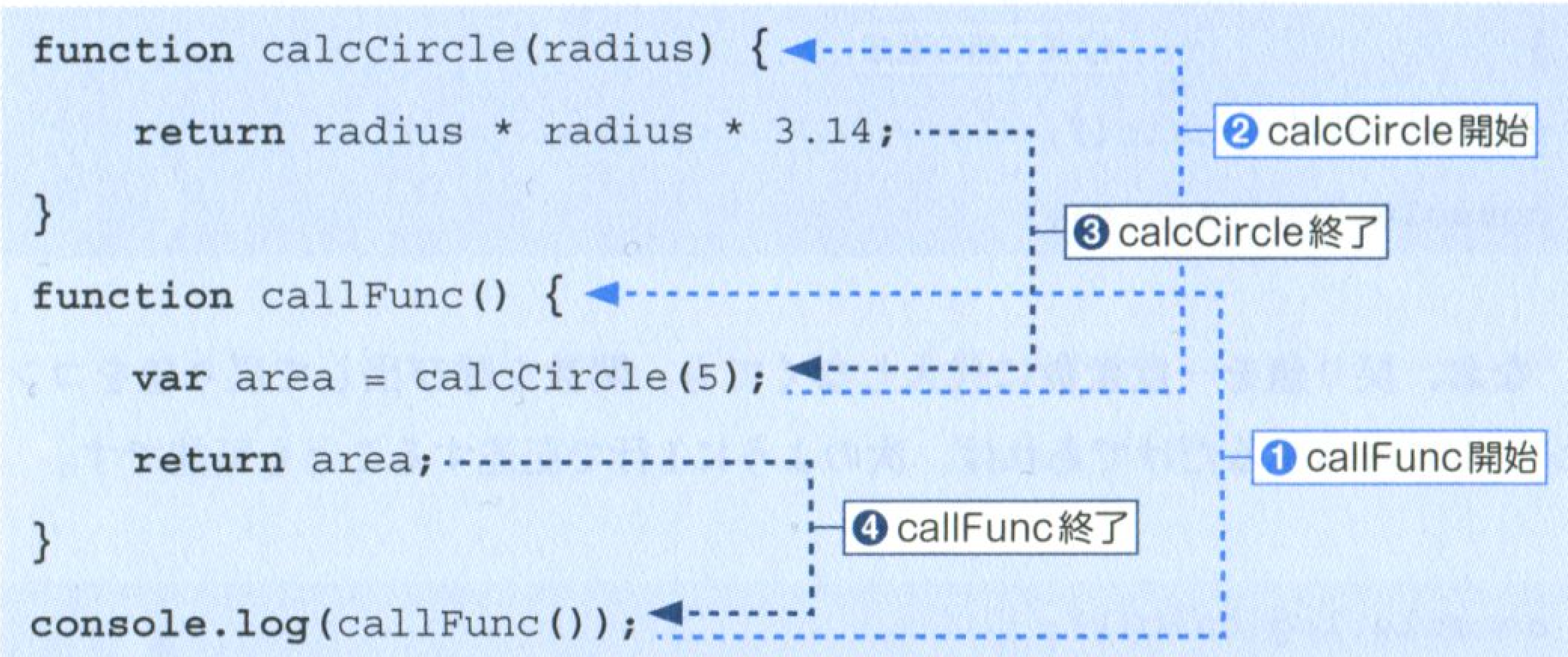
リスト6.4 引数と戻り値がある関数の利用

```
function calcCircle(radius) {
    return radius * radius * 3.14;
}
function callFunc() {
    var area = calcCircle(5);
    return area;
```

```
}
console.log(callFunc());
```

■実行結果

```
78.5
```



実行イメージ

Point!

- 関数は引数を受け取り、戻り値を返す。
- 関数は定義しただけでは実行されない。呼び出しを行なったタイミングで実行される。



■ Check Test 1

Q1 以下は、受け取った引数をコンソールに表示する関数です。この関数を呼び出し、コンソールに"ABC"と表示するにはどのような文を記述すれば良いですか？

```
function func(x) {
  console.log(x);
}
```

Q2 ある関数内で、変数xの値を戻り値として返す文はどのように記述しますか？

➲解答は巻末に掲載

6.2 組み込み関数

組み込み関数

前節では、関数を自分で定義する方法について学びましたが、JavaScriptにあらかじめ用意されている（組み込まれている）関数もあります。このようにJavaScriptがすでに備えている関数のことを**組み込み関数**と呼びます。JavaScriptには数多くの便利な組み込み関数がありますが、ここではよく使うものを紹介します（表6.1）。

表6.1 組み込み関数一覧

関数名	説明
parseInt	文字列型の数字列を数値型（整数）に変換する
parseFloat	文字列型の数字列を数値型（実数）に変換する
isNaN	指定された値がNaNであるかどうか調べる
encodeURIComponent	指定された文字列をURLエンコードする
decodeURIComponent	指定された文字列をURLデコードする

用語解説

【URLエンコード／デコード】

URLとして使用できない文字を、特殊文字に変換することを**URLエンコード**と呼びます。URLエンコードされた文字を元の状態に復元することを**URLデコード**と呼びます。

数値変換に関する関数

文字列型の値を数値型の値に変換する関数2種と、数値の演算結果を調べる関数を紹介します。

最初に**parseInt**（パースイント）**関数**の利用例を示します。parseIntは、引数として指定された文字列を整数型に変換し、戻り値として返す関数です。

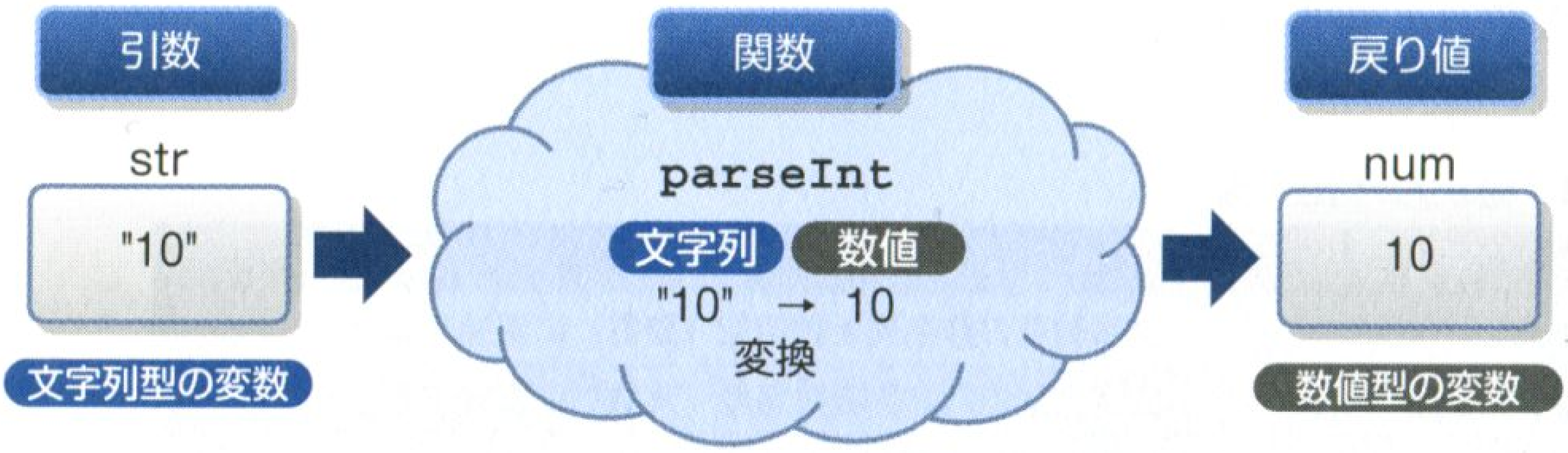
リスト6.5 parseIntの利用

```
var str = "10";
console.log(str + 10);
var num = parseInt(str);
console.log(num + 10);
```

■実行結果

```
1010
20
```

リスト6.5では、文字列型の"10"を数値型の10に変換して変数numに格納しています。このとき、変数strの値は文字列型のままであることに注意してください。



リスト6.5の実行イメージ

なお、引数に指定する文字列が小数点を含む場合は、実数型に変換する**parseFloat**（パースフロート）**関数**を使います。利用方法はparseInt関数と同じです。

parseInt関数やparseFloat関数は、アルファベットなどの数値に変換できない値が引数として指定された場合に戻り値として**NaN**（ナン）を返します。NaNは、演算結果が数値にならなかったことを表わす特殊な値です。

結果がNaNになってしまった場合、演算失敗なのでエラー処理を行なう必要があります。値がNaNであるかどうかを調べるには**isNaN**（イズナン）**関数**を使います。

リスト6.6 isNaNの利用

```
var str = "A";
var num = parseInt(str);
if(isNaN(num)) {
    console.log("変換失敗");
}
```

■実行結果

```
変換失敗
```

isNaN関数は、引数として指定された値がNaNであった場合にtrueを返します。リスト6.6では、アルファベットをparseInt関数で数値型に変換しようとしているため、変換に失敗して変数numにNaNが代入されます。

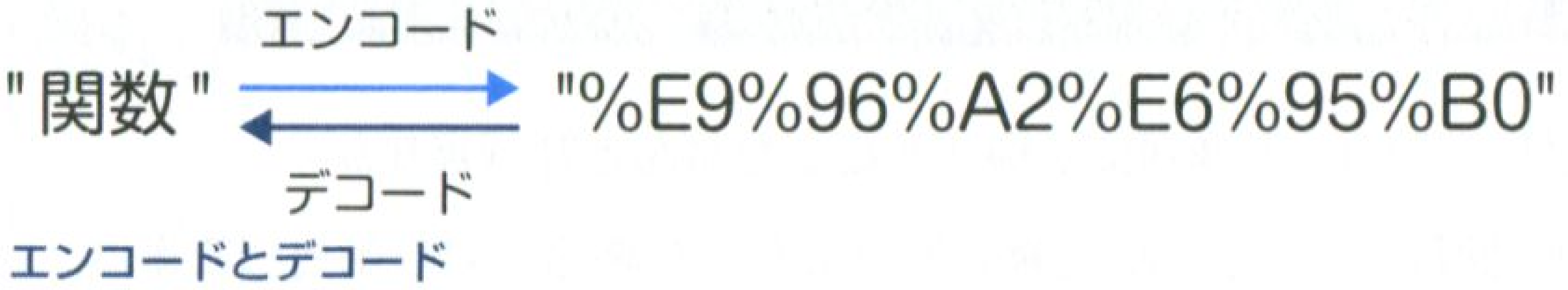
エンコード／デコードに関する関数

Web上で文字列を送信する場合、全角文字や一部の記号はそのままの状態では送信できないため、送信可能な状態に変換します。この作業を**URLエンコード**と呼びます。

試しに、Yahoo!検索で「関数」と入力して検索すると、ブラウザのアドレスバーに以下のような文字列が表示されます。

```
http://search.yahoo.co.jp/search?p=%E9%96%A2%E6%95%B0& ⏎
search.x=1&fr=top_ga1_sa&tid=top_ga1_sa&ei=UTF-8&aq=&oq= ⏎
%E9%96%A2%E6%95%B0
```

色が付いた箇所がエンコードされた文字列です。この文字列を元に戻す（**デコード**する）と「関数」という文字列になります。一部のブラウザには全角文字がデコードした状態で表示される設定になっているものもあります。その場合はアドレスバーの文字列をコピーしてテキストエディタなどに貼り付けてみると、エンコードされた文字列が確認できるはずです。



エンコードとデコード

通常、文字列の送信にはHTMLのフォームを利用します。フォームからデータを送信する場合は自動的にURLエンコードが行なわれますが、JavaScriptで別のページにデータを送信する場合はURLエンコードが行なわれません。そこで、送信前に**encodeURIComponent関数**を使ってURLエンコードを行なう必要があるのです。

リスト6.7 encodeURIComponent、decodeURIComponentの利用

```
var encode = encodeURIComponent("こんにちは");
console.log(encode);
var decode = decodeURIComponent(encode);
console.log(decode);
```

■実行結果

```
%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF
こんにちは
```


encodeURIComponent関数と**decodeURIComponent関数**は、引数に指定された文字列をエンコードまたはデコードして、変換後の文字列を戻り値として返します。この関数を使ったJavaScriptによるデータの送信方法は第13章で紹介します。

Point!

- ▶ JavaScriptにはあらかじめ便利な関数が用意されている。それを組み込み関数と呼ぶ。
- ▶ 文字列型の数字を計算処理などに使う場合は、parseIntやparseFloatで数値型に変換する。
- ▶ Web上で文字列を送信する場合はURLエンコードを行なう必要がある。



Check Test 2

Q1 ある文字を文字コード番号に変換することをなんと呼びますか？

Q2 parseInt関数で数値型への変換ができなかった場合、戻り値として返される値はなんですか？

➲解答は巻末に掲載

6 3 変数のスコープ

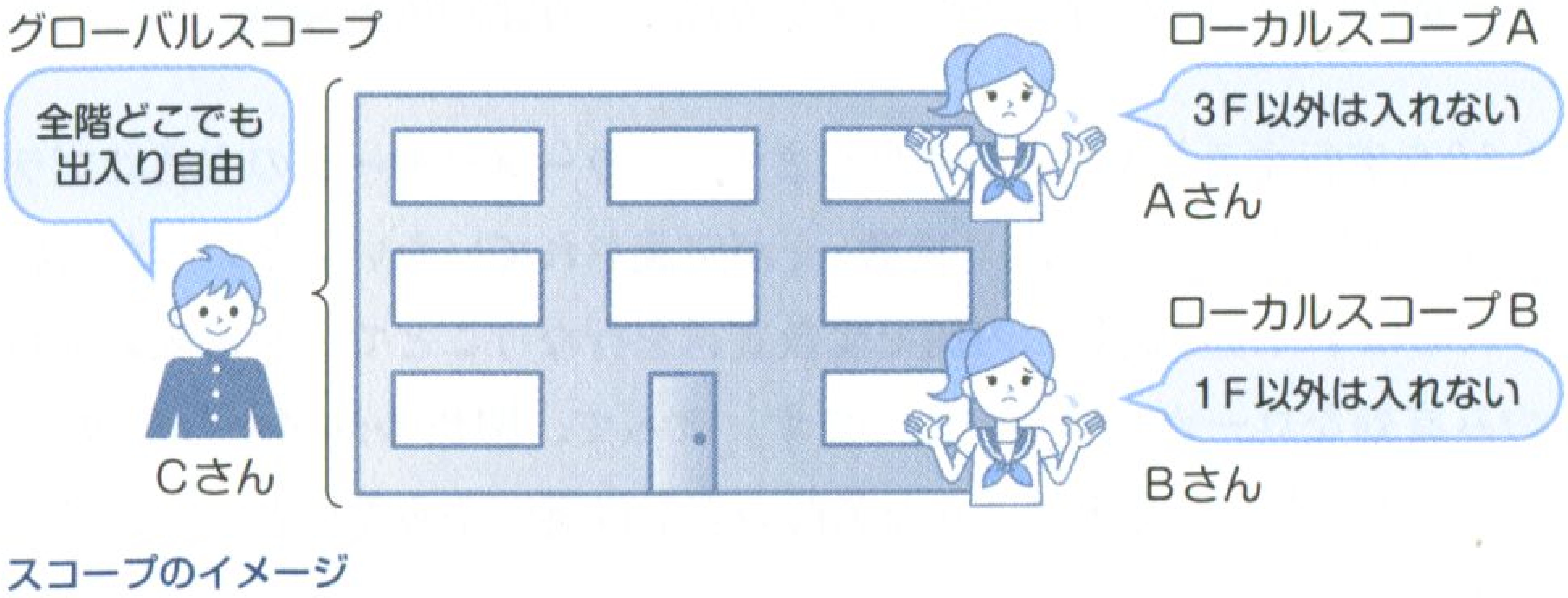
スコープとは

変数には、**スコープ**（有効範囲）が設定されています。スコープは、その変数がスクリプト中のどこからどこまでの範囲で利用できるのかを表わします。JavaScriptには、表6.2に示した2つのスコープがあり、どちらのスコープになるかは変数の宣言方法によって変わります。

表6.2 JavaScriptのスコープ

種類	説明	対象
グローバルスコープ	スクリプト全体で変数を参照できる	・関数の外側で宣言した変数 ・宣言を省略した変数
ローカルスコープ	ある関数の中でのみ変数を参照できる	・関数の内側で宣言した変数 ・関数の引数

グローバルスコープの変数は**グローバル変数**、**ローカルスコープ**の変数は**ローカル変数**と呼びます。



スコープのイメージ

では実例を見てみましょう。

リスト6.8　グロ―バル変数の利用

```
function func() {
    str = "ABC";
    console.log(str);
}
func();
console.log(str);
```

グローバルスコープ

■実行結果

```
ABC
ABC
```

リスト6.8では、変数strの宣言を省略しています。このようにするとJavaScriptが暗黙的にグローバル変数として宣言を行なってくれます。グローバル変数はどこからでも参照できるため、関数の内側と外側のどちらからでも問題なく変数の値が出力されます。

リスト6.9　ローカル変数の利用

```
function func() {
    var str = "ABC";
    console.log(str);
}
func();
console.log(str);       // エラー：関数外ではローカル変数を扱うことができない
```

ローカルスコープ

■実行結果

```
ABC
エラーメッセージ
```

リスト6.9を実行するとエラーが発生します。エラーメッセージの文言はブラウザによって多少異なりますが、「変数strが定義されていない」といった内容が表示されるはずです。関数の内側で変数宣言を行なうことで、その変数は関数内でのみ有効なローカル変数となります。よって、関数の外側から変数strの値を出力しようとしてもそのような名前の変数は見つからないという結果になるのです。

スコープの注意点と使い分け

では、異なるスコープで同名の変数を使うとどうなるでしょうか。

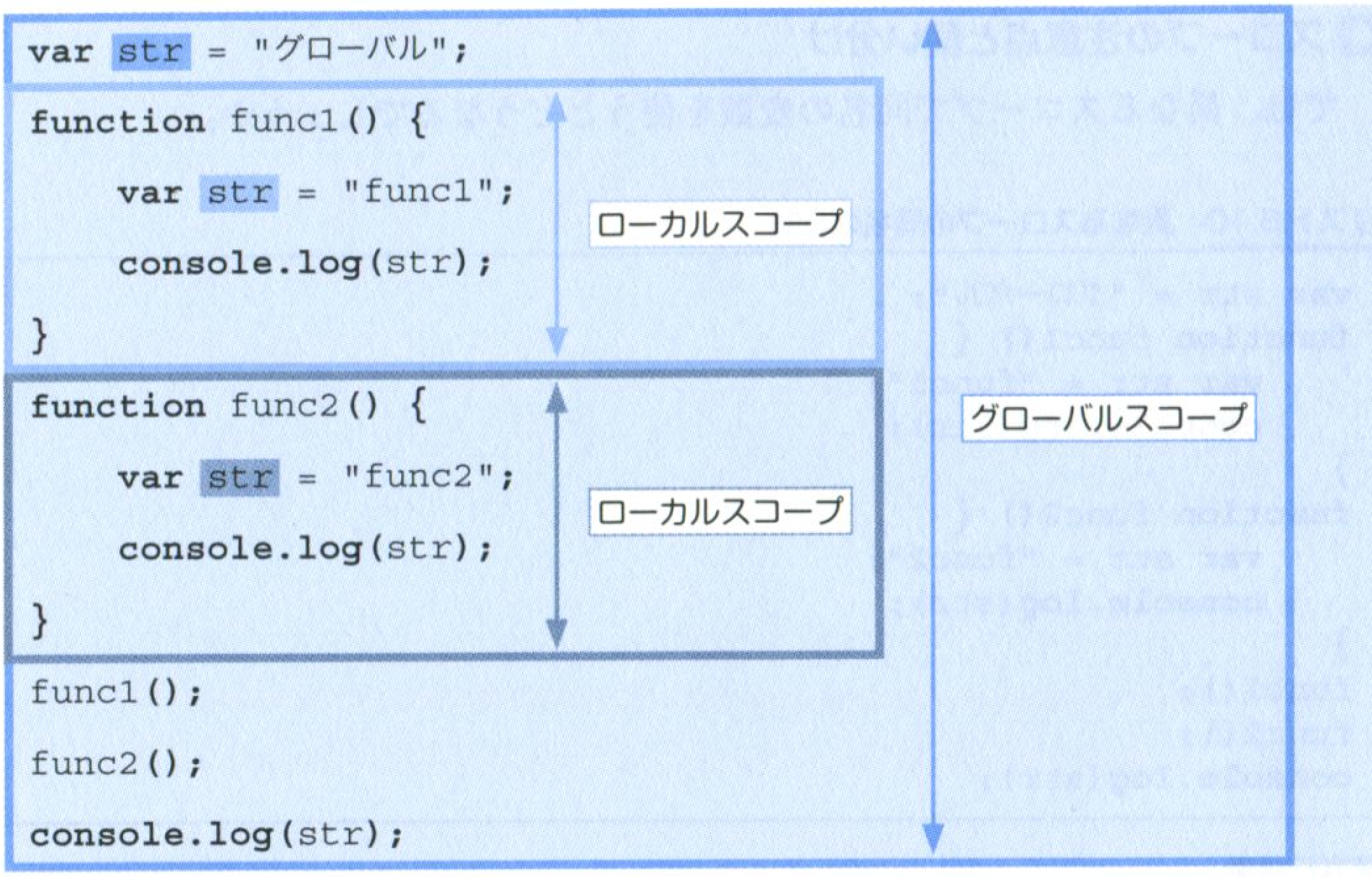
リスト6.10 異なるスコープの同名の変数

```
var str = "グローバル";
function func1() {
    var str = "func1";
    console.log(str);
}
function func2() {
    var str = "func2";
    console.log(str);
}
func1();
func2();
console.log(str);
```

■実行結果

```
func1
func2
グローバル
```

リスト6.10では、グローバルスコープと、func1のローカルスコープ、func2のローカルスコープにそれぞれstrという名前の変数が存在しています。名前が同じであってもスコープが異なる場合はまったく別の変数として扱われます。よって、3つのstrはすべて異なる変数であるということになります。func1関数とfunc2関数からはグローバル変数のstrとローカル変数のstr両方が参照できますが、両スコープに同名の変数がある場合はローカル変数のほうが優先されます。



リスト6.10の変数のスコープ

また、ローカル変数は関数が呼び出されるたびに値が初期化されてしまいますが、グローバル変数はページが更新されない限り永続的に値を持ち続けることができます。

リスト6.11　グローバル変数の値の変移

```
var global = 0;
function func() {
    global++;
}
for(var i=0; i < 10; i++) {
    func();
}
console.log(global);
```

■実行結果

```
10
```

リスト6.11では、func関数内で変数globalをインクリメントしています。グローバル変数の値は保持され続けるので、func関数が呼び出されるたびに変数globalの値が増えていきます。

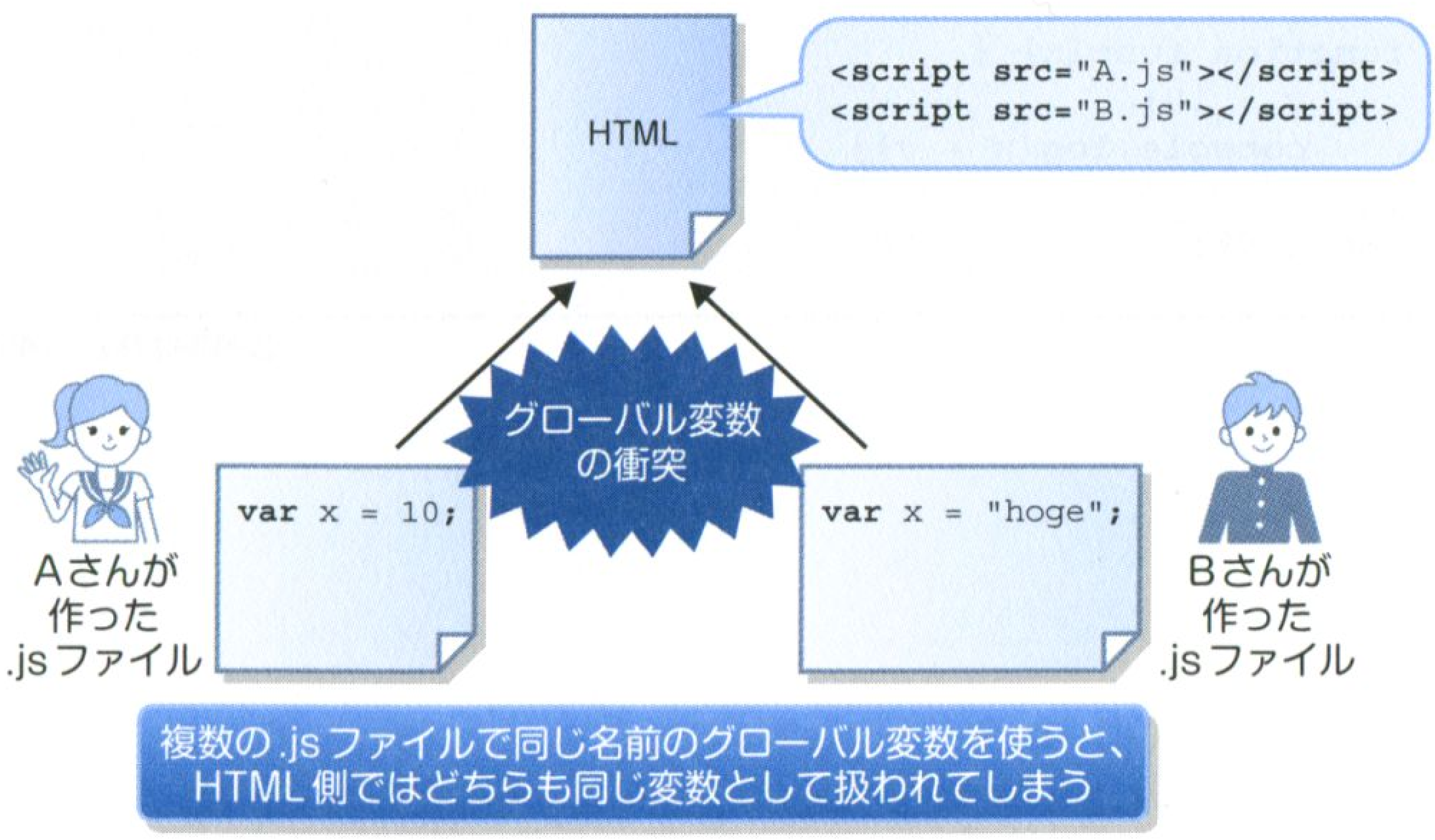
```
var global = 0;
function func() {
    global++;
}
for(var i=0; i < 10; i++) {
    func();
}
console.log(global);
```

10回呼び出される

変数globalは
グローバルスコープ内で
値が保持され続ける

値の変移のイメージ

スコープの使い分けですが、基本的にはローカル変数を利用するものと思ってください。グローバル変数を使うと、複数のJavaScriptファイル（.jsファイル）を読み込んだ際に変数名が衝突し、互いのスクリプトに影響を与えてしまう可能性があるため、あまり好ましくありません。複数の関数から1つの変数を扱う場合や、ページが開いている間、値を保持し続けたい場合などにはグローバル変数を使っても良いですが、必要最小限に抑えるようにしましょう。



グローバル変数の衝突

Point!

- 変数には有効範囲（スコープ）があり、その範囲内でのみ扱うことができる。
- 変数の宣言方法によってスコープが決定される。



■ Check Test 3

Q1 以下のスクリプト中の変数xとyはそれぞれグローバル変数とローカル変数のどちらですか？

```
var x = 10;
function func() {
    var y = 10;
    console.log(x + y);
}
func();
```

Q2 以下のスクリプト中の変数xとyはそれぞれグローバル変数とローカル変数のどちらですか？

```
function func(x) {
    y = 10;
    console.log(x + y);
}
func(20);
```

➲解答は巻末に掲載

6-4 関数の応用

関数の代入と匿名関数

最後に、知っていると役に立つ、一歩進んだ関数の記述方法について学びます。まずは**関数の代入**です。

関数は、数値や文字列のように変数に入れて扱うことができます。この方法を使うと、条件に応じて異なる関数を変数に設定したり、関数を引数として別の関数に渡したりと、より柔軟に関数を利用できます。

文法 関数の代入

```
変数名 = 定義済みの関数名;
```

代入の例を見てみましょう。

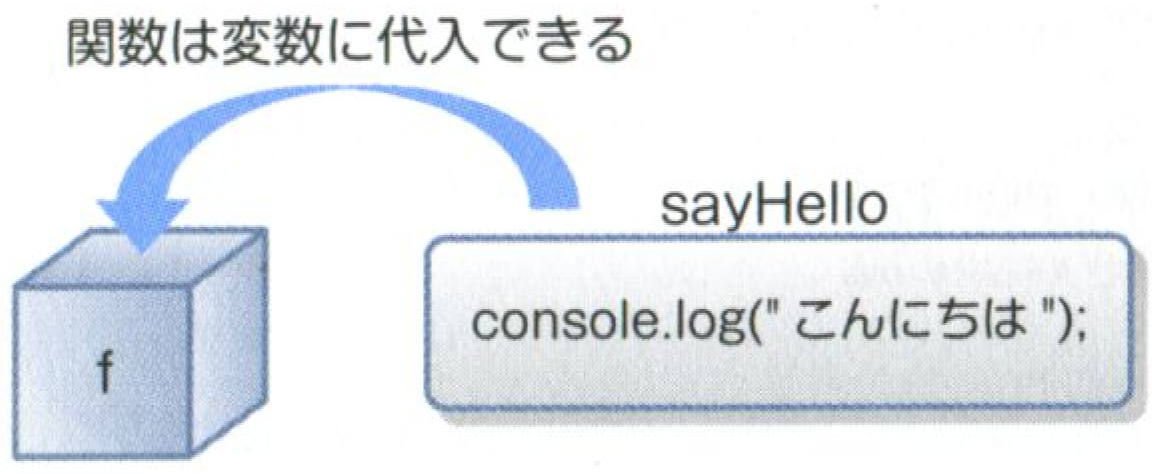
リスト6.12 関数を変数へ代入する

```
function sayHello() {
    console.log("こんにちは");
}
var f = sayHello;
f();
```

■実行結果

```
こんにちは
```

リスト6.12では、sayHello関数を定義して、変数fに代入しています。代入するときは関数名の後ろに丸かっこ()は付けません。



関数の代入

また、関数の定義と変数への代入を同時に行なうこともできます。このような式を**関数式**と呼びます。

文法 関数式

```
変数名 = function 関数名() { };
```

文末にセミコロンが必要！

関数式は、以下のように使います。

リスト6.13 関数の定義と変数への代入を同時に行なう

```
var f = function sayHello() {
    console.log("こんにちは");
};
f();
```

■実行結果

```
こんにちは
```

リスト6.13のような記述を行なうときに気をつけなければならないのは、関数定義ブロックの閉じかっこ（}）の後ろにセミコロン（;）を付けることです。どこまでが変数fに代入する値なのかを明確にしましょう。次のように、変数に代入する値が数値であろうと関数であろうと、同じように記述すると理解しておけば良いでしょう。

```
// 数値の代入
var f = 10;
// 関数の代入
var f = function sayHello() {
        console.log("こんにちは");
};
```

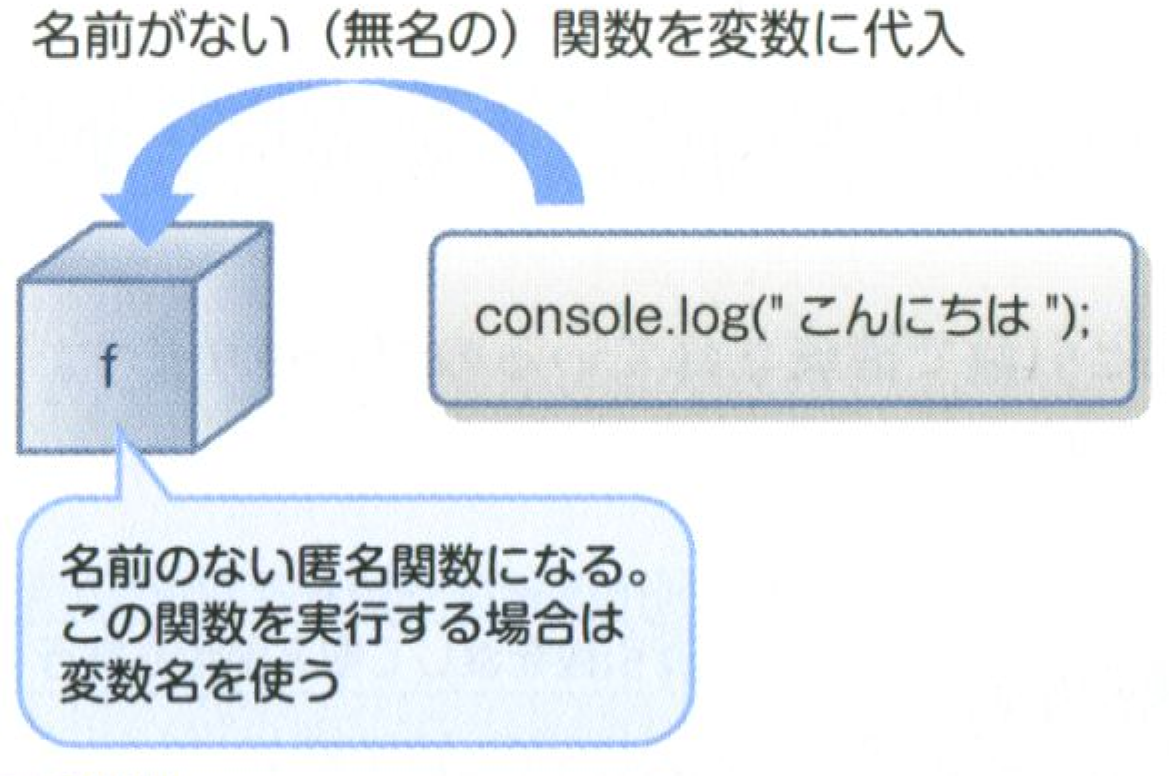
リスト6.13では、関数sayHelloを変数fに代入し、以降は「f」という変数名を使って関数を呼び出しています。このように関数名を使わない場面では、関数名の省略が可能となっています。関数名を省略すると、名前のない**匿名関数**(とくめい)となります。

リスト6.14 匿名関数の代入

```
var f = function() {
    console.log("こんにちは");
};
f();
```

■実行結果

```
こんにちは
```



匿名関数

即時関数

関数は定義しただけでは実行されず、呼び出しを行なって初めて動作するものでした。しかし、1回だけしか呼び出しを行なわない関数の場合、名前を付けて定義した後に呼び出しを行なう、というスクリプトをいちいち記述していると冗長なスクリプトになってしまいます。そのようなときは、定義と同時にすぐに実行される**即時関数**(そくじ)を使います。

文法 即時関数

```
(function(引数を入れる変数){処理})(引数として渡す値);
```

匿名関数の定義　定義した関数の呼び出し

記述方法は、匿名関数の定義を丸かっこ()で囲み、さらにその後ろにもう1つ丸かっこ()を付けて引数を渡します。引数を省略する場合も丸かっこ()は必ず必要です。

利用例は次のとおりです。

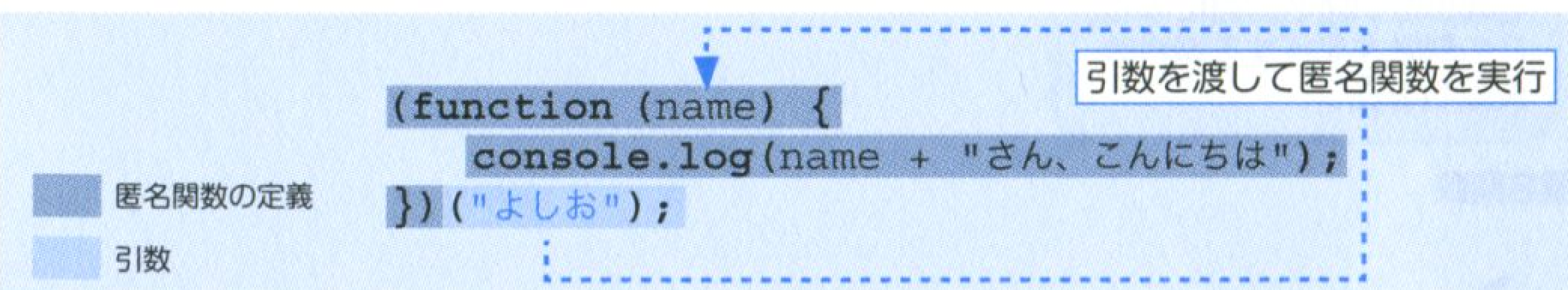
リスト6.15 即時関数の利用

```
(function (name) {
    console.log(name + "さん、こんにちは");
})("よしお");
```

■実行結果

```
よしおさん、こんにちは
```

スクリプト全体が匿名関数の定義と引数で構成されているのがわかります。



匿名関数の構成

Point!

- ▶ 記述方法次第で名前のない関数やすぐに実行される関数を作ることができる。
- ▶ 関数の定義文を値として扱う記述方法を関数式と呼ぶ。



この章の到達度チェック

Q1 以下の関数は、三角形の底辺と高さを引数として受け取り、面積を戻り値として返します。空欄を埋め、底辺が30、高さが50の三角形の面積を求める関数を完成させてください。

```
function calcTriangle(base, height) {
    return base * height / 2;
}
var area = [            ];
console.log(area);
```

Q2 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
var n = parseFloat("10.5");
if(!isNaN(n)){
    console.log("OK");
} else {
    console.log("NG");
}
```

Q3 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
var x = 10;
function plus() {
    x += 20;
}
plus();
console.log(x);
```

Q4 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
var x = 10;
function plus(x) {
    x += 20;
}
plus(x);
console.log(x);
```

Q5 空欄を埋め、func関数を変数fに代入するスクリプトを完成させてください。

6

```
function func() {
    console.log("ABC");
}
var f = [          ];
```

Q6 以下の関数を呼び出し、コンソールに"ABC"と表示するには、どのような文を記述すれば良いですか？

```
var f = function (x) {
    console.log(x);
};
```

Q7 空欄を埋め、コンソールに"こんにちは"と表示する関数を実行するスクリプトを完成させてください。

```
(function (str) {
   console.log(str);
}) [          ];
```

➲解答は巻末に掲載

第7章

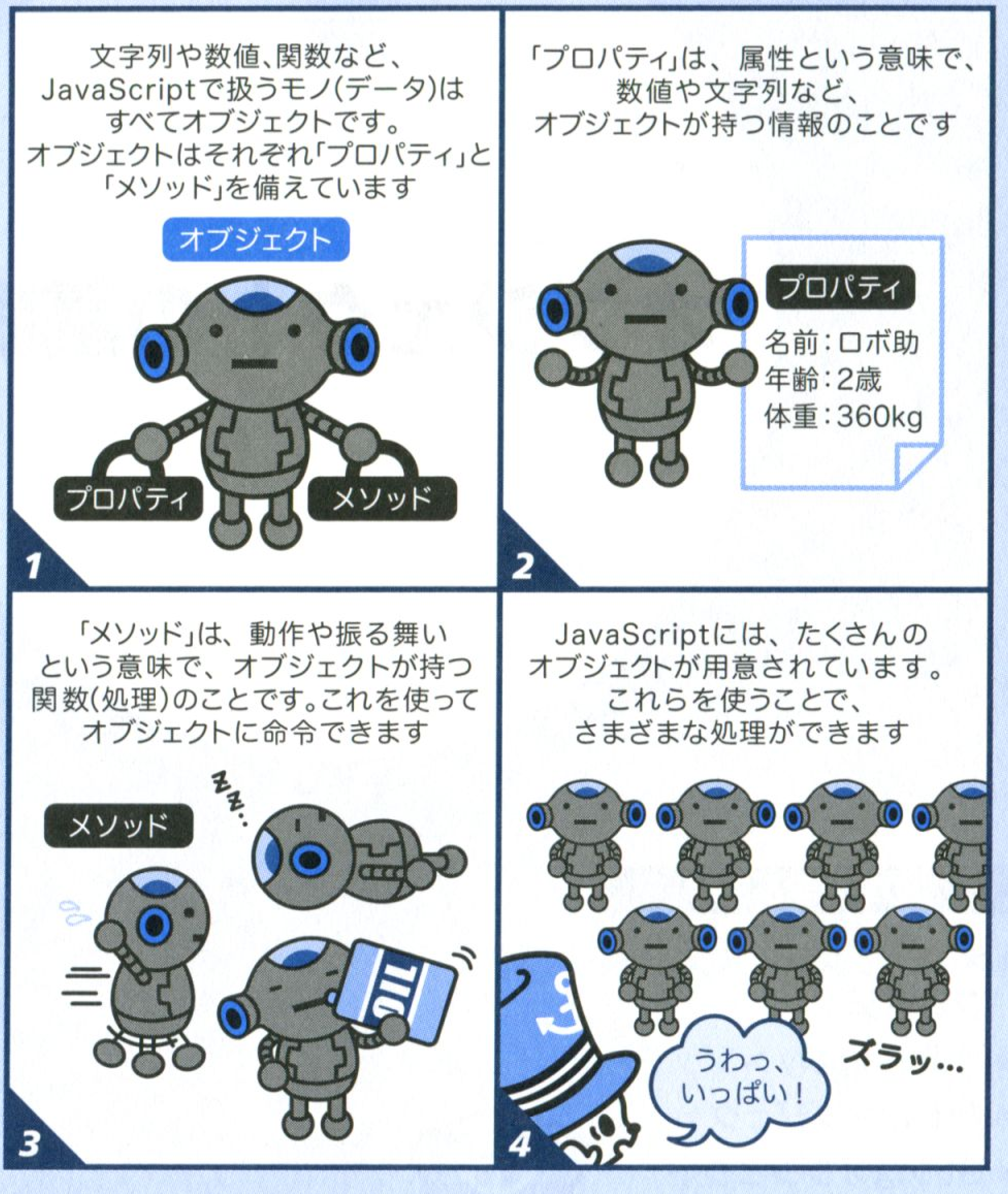
オブジェクトの基礎

この章で学ぶこと

- オブジェクト、メソッド、プロパティ
- call ／ applyメソッド
- Arrayオブジェクト
- Stringオブジェクト
- 正規表現



オブジェクトは「プロパティ」と「メソッド」を持っている

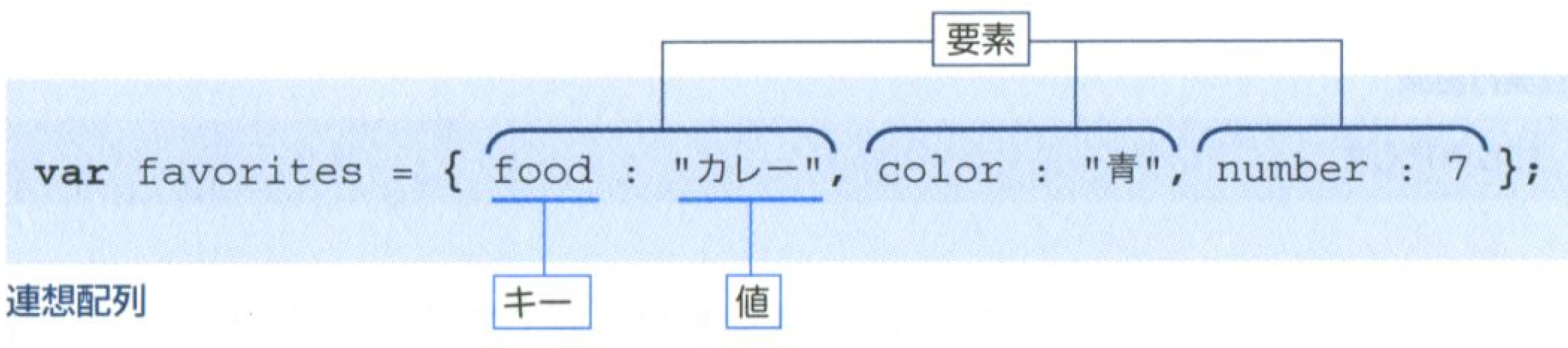


オブジェクトはJavaScriptの要です。オブジェクトの基礎をしっかりと理解し、JavaScriptのオブジェクトを効果的に活用できるようになりましょう。

7-1 オブジェクト、メソッド、プロパティ

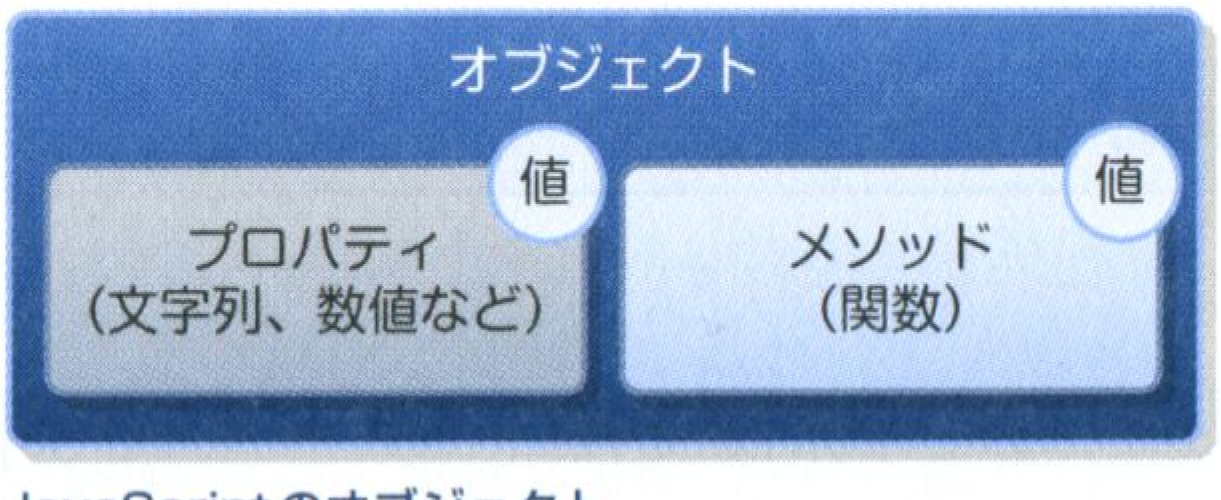
オブジェクトとは

文字列や数値、関数など、JavaScriptで扱うデータは、実はすべて**オブジェクト**です。もっと具体的に言うと、JavaScriptのオブジェクトとは、キーと値の集合体のことです。第4章で学んだ連想配列を思い出してください。これはオブジェクトの別名です。



連想配列

数値や文字列などを値として持つ要素は**プロパティ**、関数を値として持つ要素は**メソッド**と呼ばれます。



JavaScriptのオブジェクト

> **Note** 結局のところ、メソッド＝関数です。オブジェクトでは関数のことをメソッドと呼ぶ、と考えると理解しやすいでしょう。

ではオブジェクトの作成例を見てみましょう。

リスト7.1　オブジェクトの作成

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   var obj = {
       name: "よしお",
       greet: function() { console.log("はじめまして"); }
   };
   obj.greet();
  </script>
 </head>
 <body>
 </body>
</html>
```

要素の値として関数を指定

■実行結果

```
はじめまして
```

記述方法は基本的に第4章で学んだ連想配列と同じです。要素の値として関数を指定することでメソッドを作成できます。このとき、関数名が必要なことはほとんどないので匿名関数にして良いでしょう。メソッドを呼び出すには、関数名の代わりにキーを指定します。

文法 メソッドの作成

```
var オブジェクト名 = { キー : 関数,・・・ };
```

文法 メソッドの呼び出し

```
var オブジェクト名.キー
```

Note
以降のサンプルコードでは、JavaScript（リスト7.1の網掛け部分）のみを掲載します。

thisキーワード

オブジェクトの中では、this（ディス）というキーワードは自分自身のオブジェクトのことを指します。あるメソッドの中で、自身が所属するオブジェクトが持つ別のメソッドやプロパティを利用したい場合、先頭に「this.」を付けて使います。

リスト7.2　オブジェクトの中でのthisキーワード

```
var obj = {
    name: "よしお",
    greet: function() {
        console.log(this.name + "です、はじめまして");
    }
};
obj.greet();
```

■実行結果

```
よしおです、はじめまして
```

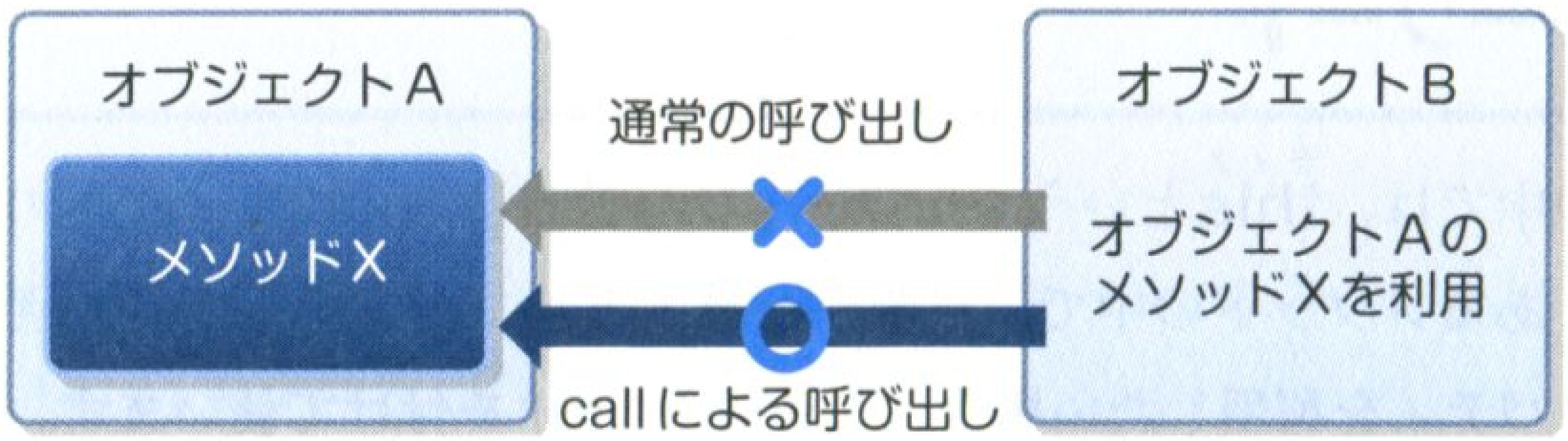
リスト7.2では、thisキーワードが表わすのはobjオブジェクトなので、this.nameはobj.nameと記述したのと同じことになります。

```
var obj = {
    name: "よしお",
    greet: function() {
        console.log(this.name + "です、はじめまして");
    }
};
obj.greet();
```

thisはobjを表わす

別のオブジェクトのメソッドを利用する

メソッドはオブジェクトの所有物なので、通常、他のオブジェクトが使う（呼び出す）ことはできません。そこで、あるオブジェクトから別のオブジェクトのメソッドを呼び出すcall（コール）メソッドという特殊なメソッドが用意されています。



別のオブジェクトのメソッドを呼び出す

callは「別のオブジェクトのメソッドを借りて、自分のメソッドのように扱う」ことができるとても便利なメソッドです。すべてのメソッドが、このcallメソッドを利用できます。

文法 callメソッド

```
メソッドを所有するオブジェクト.メソッド名.call(メソッドを借りるオブジェクト);
```

まずはリスト7.3を見てください。

リスト7.3 callメソッドの利用

```
var obj1 = {
    name: "よしお",
    greet: function() {
        console.log(this.name + "です、はじめまして");
    }
};
var obj2 = {
    name: "ユリ子"
};
obj1.greet.call(obj2);
```

■実行結果

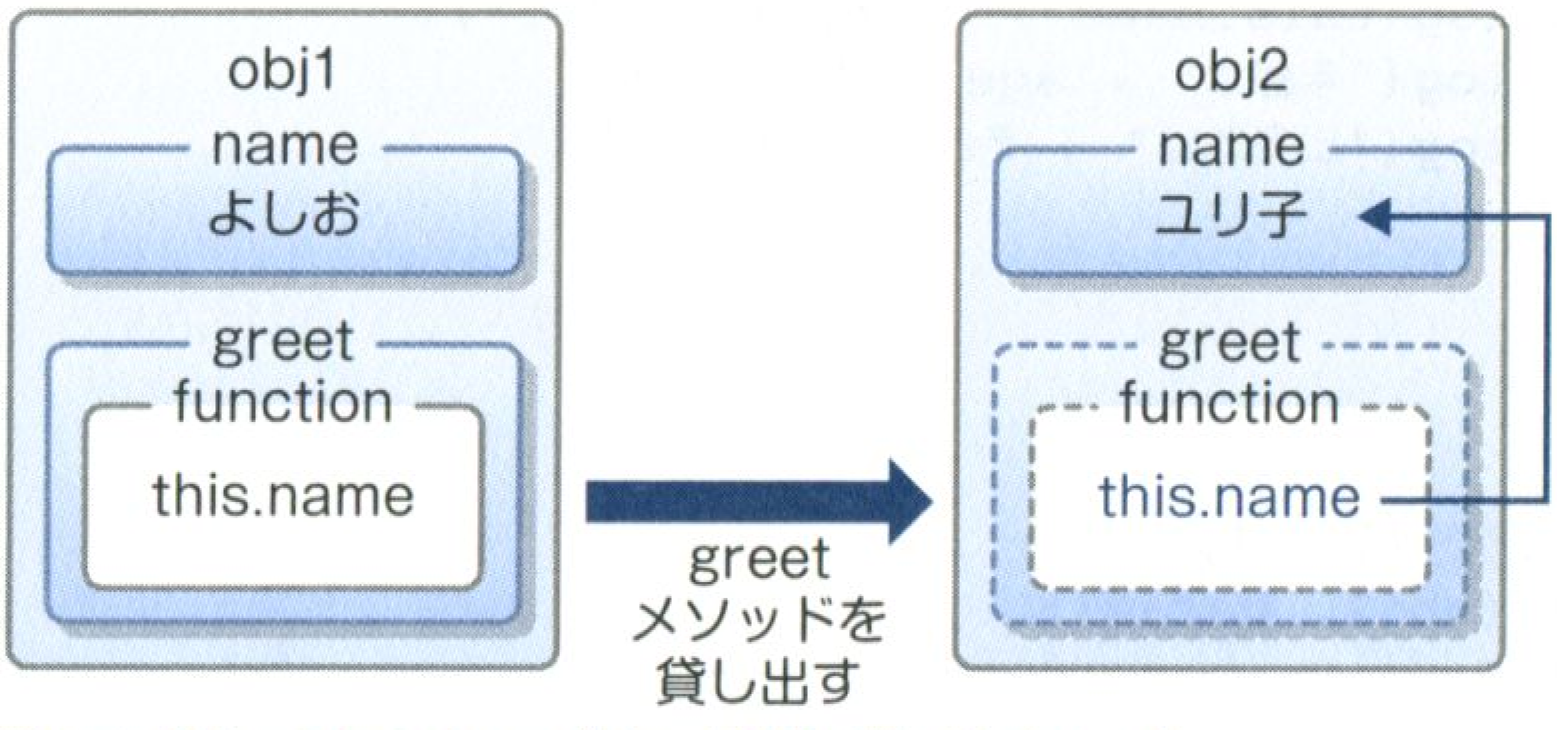
```
ユリ子です、はじめまして
```

ユリ子さんは、「よしおさんが持っているメソッドを自分も利用したい」ということでcallメソッドを使っています。するとあたかもユリ子さんが自身のメソッドを呼び出したかのようにgreetメソッドが実行されます。このとき、greetメソッド内のthisキーワードが表わすのはユリ子さん(obj2)になります。

```
obj1.greet.call(obj2);
```

obj1：メソッドの所有者
obj2：メソッドを借りる人

メソッドを借りている間は所有権も借りている人に移る、といったイメージです。



別のオブジェクトのメソッドを一時的に借りるイメージ

借りるメソッドに引数がある場合は、callメソッドの第2引数（2つ目の引数）以降に指定します。

リスト7.4　引数のあるメソッドをcallメソッドで呼び出す

```
var obj1 = {
    name: "よしお",
    greet: function(age, from) {
        console.log(this.name + "です、はじめまして");
        console.log("年齢は" + age + "歳です");
        console.log("出身地は" + from + "です");
    }
};
var obj2 = {
    name: "ユリ子"
};
obj1.greet.call(obj2, 6, "アメリカ");
```

■実行結果

```
ユリ子です、はじめまして
年齢は6歳です
出身地はアメリカです
```

また、callメソッドとよく似た**apply**（アプライ）メソッドもあります。使い方はほとんど同じですが、applyメソッドは第2引数以降を配列形式にして渡します。

リスト7.5　applyメソッドの利用

```
var obj1 = {
    name: "よしお",
    greet: function(age, from) {
        console.log(this.name + "です、はじめまして");
        console.log("年齢は" + age + "歳です");
        console.log("出身地は" + from + "です");
    }
};
var obj2 = {
    name: "ユリ子"
};
var param = [6, "アメリカ"];
obj1.greet.apply(obj2, param);
```

■実行結果

```
ユリ子です、はじめまして
年齢は6歳です
出身地はアメリカです
```


Point!

- オブジェクトは内部にプロパティとメソッドを持つ。
- 関数を値として持つ要素をメソッドと呼ぶ。
- メソッドはcallやapplyメソッドによって貸し借りができる。



■ Check Test 1

Q1 以下のオブジェクトが持つメソッドを実行するには、どのような命令を記述しますか？

```
var obj = {
        x: function() { console.log("こんにちは"); }
};
```

Q2 メソッドの中で記述すると、オブジェクト自身を表わすキーワードはなんですか？

➲解答は巻末に掲載

7-2 組み込みオブジェクト

組み込みオブジェクトとは

関数と同じくオブジェクトの場合も、JavaScriptがあらかじめ用意しているオブジェクトがあります。これを**組み込みオブジェクト**と呼びます。JavaScriptには、さまざまな種類の組み込みオブジェクトがあります。

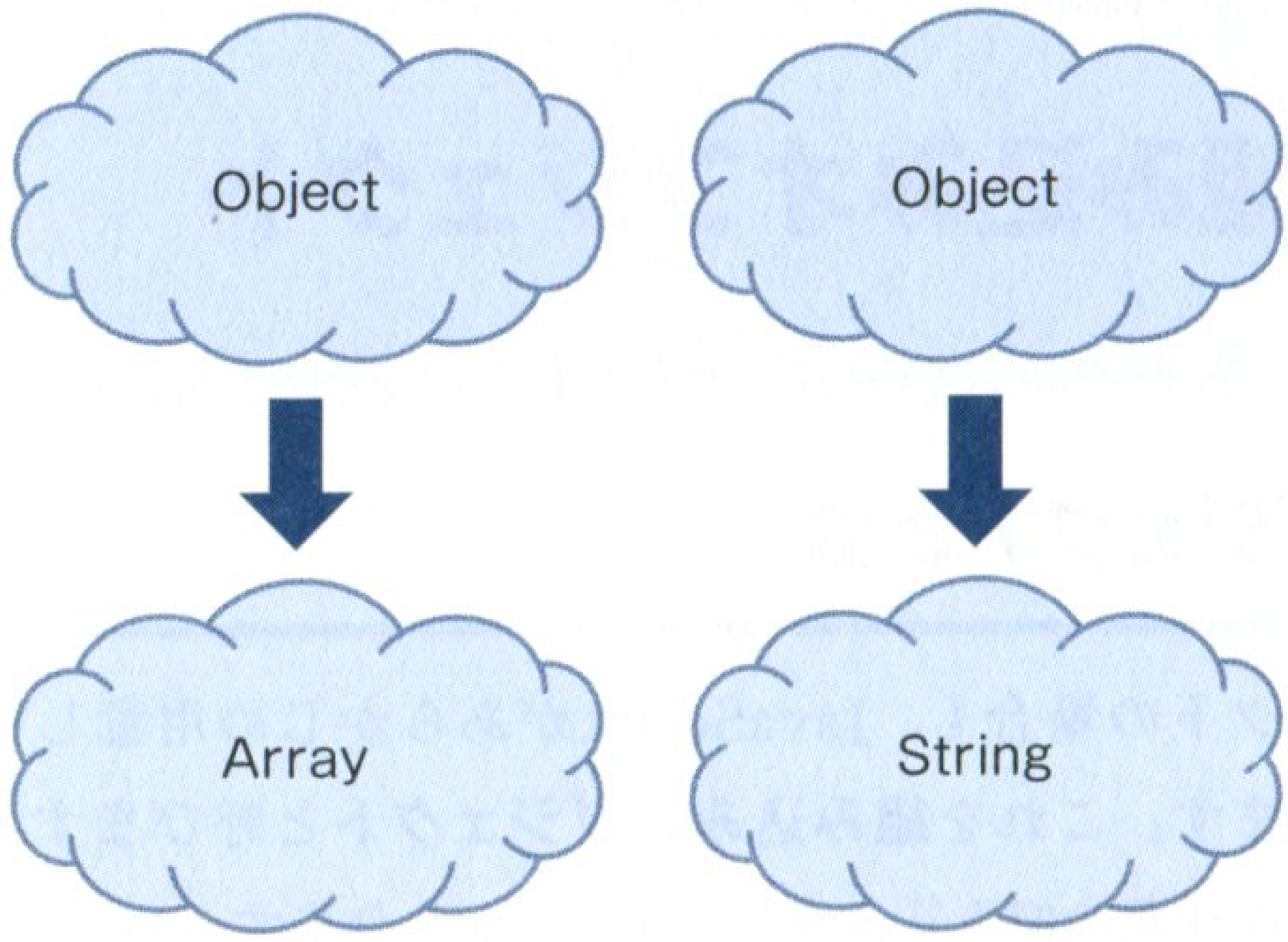
表7.1 組み込みオブジェクト一覧

オブジェクト名	説明
Object	すべてのオブジェクトの元となるオブジェクト
Date	日時操作機能を持つオブジェクト
Math	計算機能を持つオブジェクト
Array	配列操作機能を持つオブジェクト
String	文字列操作機能を持つオブジェクト

以降では、この中でも特によく使われるオブジェクトについて、利用方法の一例を見ていきましょう（ここで取り上げたもの以外にもさまざまな使い方があります）。

Objectオブジェクト

波かっこ{}で初期化したオブジェクトは、**Object（オブジェクト）オブジェクト**になります。よって、連想配列は、Objectオブジェクトです。すべてのオブジェクトは、このObjectオブジェクトを元にして作られています。Objectオブジェクトが持つプロパティとメソッドはすべてのオブジェクトで利用することが可能です。



すべてのオブジェクトはObjectを拡張して作られている

Dateオブジェクト

Date（デイト）オブジェクトは、日時をミリ秒形式のデータとして持つオブジェクトです。日付同士の比較を行なう場合などに利用します。Dateオブジェクトには、表7.2のメソッドがあります。

表7.2　Dateオブジェクトのメソッド

メソッド名	説明
getFullYear	オブジェクトが持つ日時データの「年」を返す
getMonth	オブジェクトが持つ日時データの「月」を返す。Date オブジェクトは、月のみ 0 ～ 11 までの数値で管理されているため、getMonth メソッドで取得した値を実際の月に直すには 1 加算する必要がある
getDate	オブジェクトが持つ日時データの「日」を返す
getHours	オブジェクトが持つ日時データの「時間」を返す
getMinutes	オブジェクトが持つ日時データの「分」を返す
getSeconds	オブジェクトが持つ日時データの「秒」を返す

Dateオブジェクトの使い方は次のとおりです。

文法 Dateオブジェクトの生成

```
var オブジェクト名 = new Date(年, 月, 日);
```

引数で指定された年月日を持つオブジェクトが生成されます。月には実際の月から-1した値を指定してください。引数を省略した場合は、現在の日時をデータとして持ちます。ここでいう現在の日時とは、記述したJavaScriptが動作するパソコン（クライアントPC）の日時のことです。

年／月／日は、表7.3のメソッドを使って取得できます。

文法 Dateオブジェクトのメソッドの記述

```
Dateオブジェクト.メソッド名()
```

リスト7.6　Dateオブジェクトの利用

```
var today = new Date();
console.log("今日は" +
            today.getFullYear() + "年" +
            (today.getMonth() + 1) + "月" +
            today.getDate() + "日です。");
```

■実行結果

```
今日は2012年12月20日です。
```

Mathオブジェクト

Mathオブジェクトは、さまざまな計算機能を提供するオブジェクトです。表7.3のメソッドがあります。

表7.3　Mathオブジェクトのメソッド

メソッド名	説明
ceil	小数点以下の値を切り上げた結果を返す
floor	小数点以下の値を切り捨てた結果を返す
round	小数点以下の値を四捨五入した結果を返す

Mathオブジェクトのメソッドは、オブジェクトを生成しなくても利用できます。

文法 Mathオブジェクトのメソッド

```
Math.メソッド名(数値);
```

リスト7.7　Mathオブジェクトの利用

```
console.log(Math.ceil(1.5));    // 切り上げ
console.log(Math.floor(1.5));   // 切り捨て
console.log(Math.round(1.5));   // 四捨五入
```

■実行結果

```
2
1
2
```

Arrayオブジェクト

Array（アレイ）オブジェクトは、通常の配列のことで、配列操作に関する機能を持っています。角かっこ[]で初期化したオブジェクトは、Arrayオブジェクトとなります。

主なプロパティとメソッドについて見ていきましょう。

◆ lengthプロパティ

配列の要素数を表わします。利用例は第4章(99ページ)を参照してください。

◆ pushメソッド

配列に要素を追加します。第4章ではインデックスを指定して要素を追加しましたが、pushメソッドを利用すると配列の末尾に要素が追加され、自動的にインデックスが割り振られます。一度に複数の要素を追加することも可能です。

文法　pushメソッド

```
配列名.push(要素1, 要素2 ・・・);
```

リスト7.8　pushメソッドの利用

```
var numbers = [10, 20, 30];
numbers.push(40, 50);
console.log(numbers);
```

■実行結果

```
[10, 20, 30, 40, 50]
```

◆ joinメソッド

配列の各要素を連結して1つの文字列を生成します。要素間の区切り文字を引数で指定します。

文法 joinメソッド

```
配列名.join("区切り文字");
```

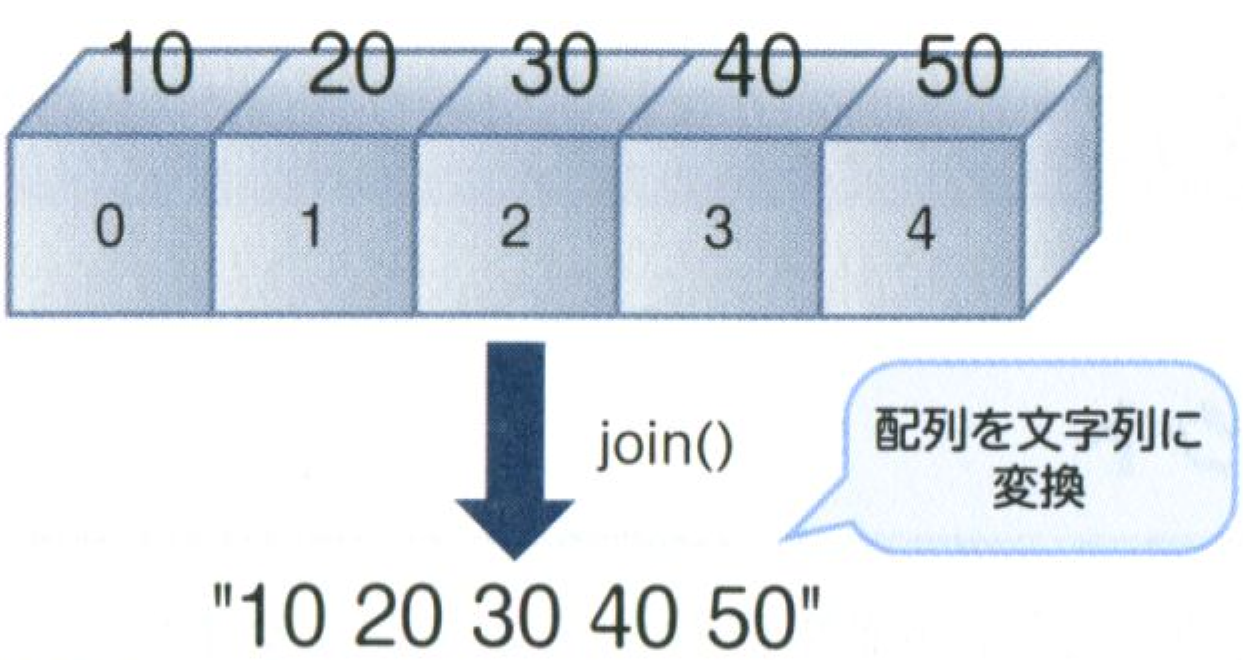
戻り値 連結した文字列

リスト7.9 joinメソッドの利用

```
var numbers = [10, 20, 30, 40, 50];
var str = numbers.join(" ");   // 半角スペースを区切り文字として要素を連結
console.log(str);
```

■実行結果

```
10 20 30 40 50
```



実行イメージ

◆ reverseメソッド

配列の要素を逆順に入れ替えます。

文法 reverseメソッド

```
配列名.reverse();
```

リスト7.10　reverseメソッドの利用

```
var numbers = [10, 20, 30, 40, 50];
numbers.reverse();
console.log(numbers);
```

■実行結果

```
[50, 40, 30, 20, 10]
```

◆ sortメソッド

配列の要素をアルファベット順に並べ替えます。

文法　sortメソッド

```
配列名.sort();
```

リスト7.11　sortメソッドの利用

```
var words = ["one", "two", "three", "four", "five"];
words.sort();
console.log(words);
```

■実行結果

```
["five", "four", "one", "three", "two"]
```

Stringオブジェクト

String（ストリング）オブジェクトは、文字列の操作機能を提供するオブジェクトです。ダブルクォートやシングルクォートで囲んだ文字列は、Stringオブジェクトのメソッドを利用できます。

主なメソッドについて見ていきましょう。

◆ indexOfメソッド、sliceメソッド

indexOfメソッドは、指定した文字列が何文字目に出現するかを調べます。先頭文字は0番目として数えます。

文法 indexOfメソッド

```
文字列.indexOf("検索文字列");
```
戻り値 検索文字列の出現位置

sliceメソッドは、文字列の中から一部を抜き出します。抜き出す文字列の開始位置と終了位置を引数で指定します。終了位置は、実際に抜き出す位置よりも1大きい値を指定します。終了位置を省略すると一番最後の文字までを抜き出します。

文法 sliceメソッド

```
文字列.slice(開始位置, 終了位置);
```
戻り値 抽出文字列

indexOfメソッドとsliceメソッドを組み合わせて使うと、文字列の中から指定部分だけ抜き出すことができます。

リスト7.12　indexOfメソッドとsliceメソッド

```
var company = "アシアル株式会社";
var index = company.indexOf("株式会社");
var name = company.slice(0, index);
console.log(name);
```

■実行結果

```
アシアル
```

リスト7.12では、会社名から"株式会社"よりも前の名前だけを抜き出しています。まず、indexOfメソッドで"株式会社"という文字列が出現する位置を調べています。先頭文字を0番目とすると、4番目に出現するという結果が返されます。次にsliceメソッドで開始位置を0、終了位置を4として文字列を抜き出しています。結果として先頭から"株式会社"の直前までの文字列が得られます。

❶indexOfで"株式会社"の開始位置を取得

```
  0 1 2 3 4 5 6 7
"アシアル株式会社"
```

❷sliceで0～4番目の直前までを抜き出す

```
  0 1 2 3 4 5 6 7
"アシアル株式会社"
```

実行イメージ

◆ splitメソッド

指定した区切り文字で文字列を分割し、配列に変換した結果を返します。配列の最大要素数を引数で指定できます。要素数の上限が必要ない場合は第2引数を省略します。Arrayオブジェクトのjoinメソッドと逆の動きをするメソッドです。

文法 splitメソッド

文字列.**split**("区切り文字", 配列の最大要素数);

戻り値 分解された文字列が代入された配列

リスト7.13 splitメソッドの利用

```
var message = "hello my dear friend";
var words = message.split(" ");  // 半角スペースを区切り文字として文字列を分割
console.log(words);
```

■実行結果

```
["hello", "my", "dear", "friend"]
```

◆ matchメソッド

文字列の中から正規表現と一致する部分を見つけ出し、戻り値として返します。

文法 matchメソッド

文字列.**match**(正規表現);
戻り値 一致部分が代入された配列

matchメソッドと次のreplaceメソッドの利用例は次節で紹介します。

◆ replaceメソッド

文字列の中から正規表現と一致する部分を見つけ出し、指定した文字列に置き換えます。

文法 replace メソッド

文字列.**replace**(正規表現, "置換文字列");
戻り値 置換後の全体文字列

Point!

- 連想配列と呼んでいたものは、実はオブジェクトというデータ型である。
- 配列や文字列もオブジェクトの一種で、あらかじめ便利なメソッドが用意されている。



■ Check Test 2

Q1 配列はなんという種類のオブジェクトですか？

Q2 連想配列はなんという種類のオブジェクトですか？

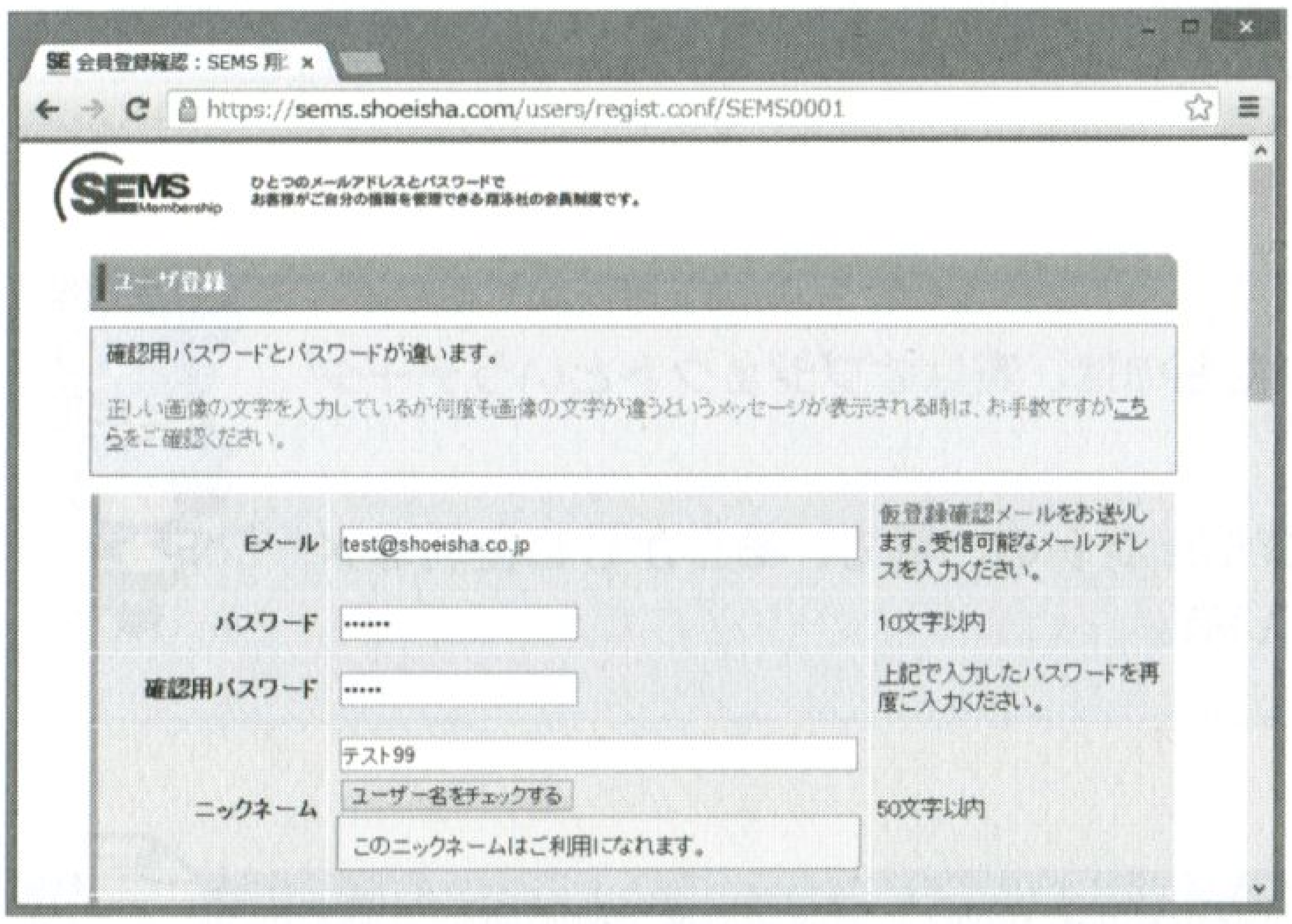
➲解答は巻末に掲載

7-3 正規表現

正規表現とは

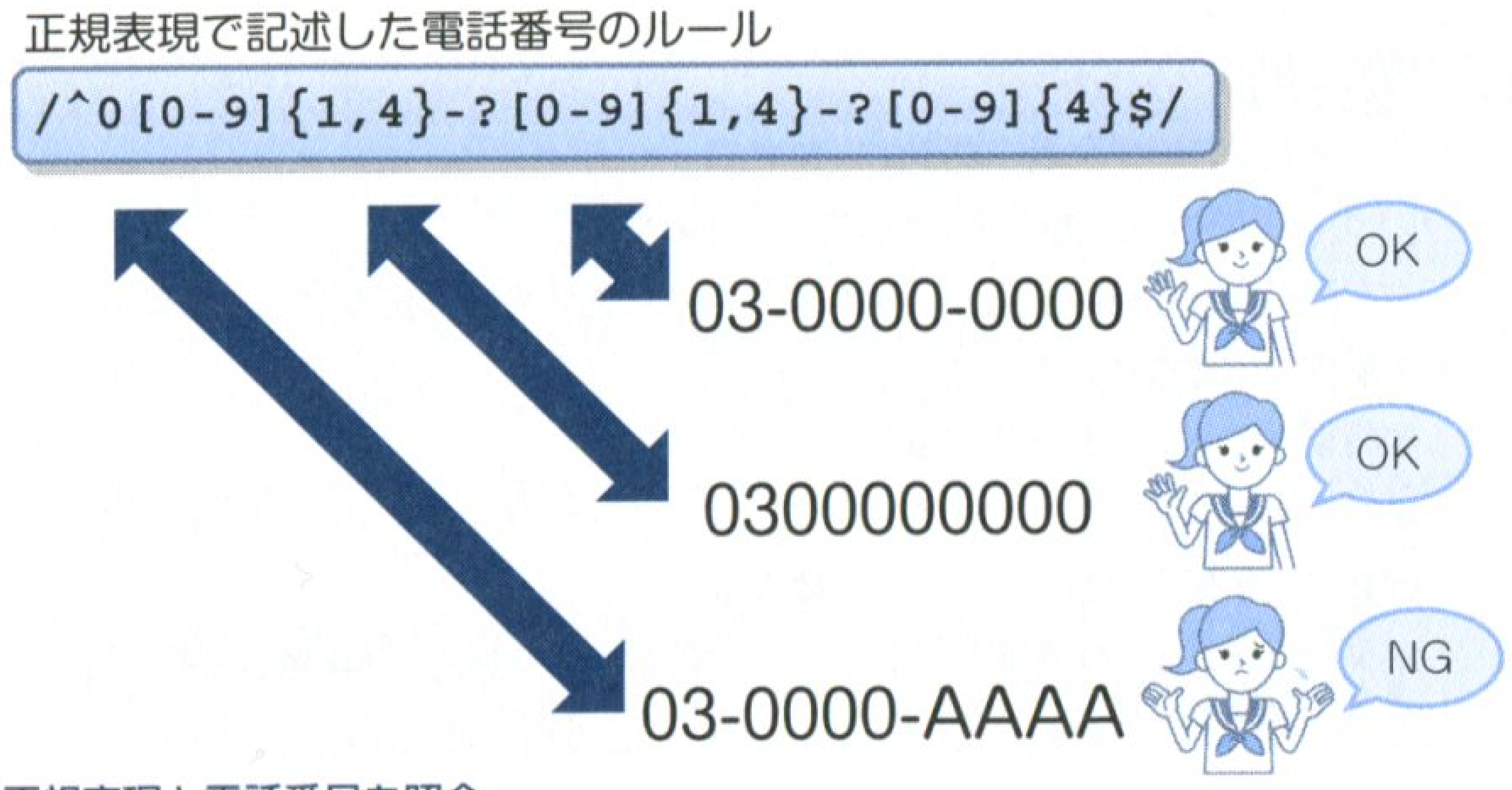
最後に、Stringオブジェクトと関係の深い**正規表現**（せいきひょうげん）という手法を取り上げます。

一般的なWebサイトでは、個人情報などの入力フォームに対して正しい形式で入力されているかどうかをチェックする処理が追加されています。



入力チェックの例

こういった入力チェック処理には、正規表現が用いられています。正規表現とは、文字列のパターンを特定するための手法のことです。たとえば、電話番号の場合、利用できる文字は数字とハイフンのみで、先頭は0、市外局番と市内局番は1～4桁、加入者番号は4桁固定、というルールがあります。このルールに沿っているかどうかを、入力された文字列と正規表現を照らし合わせることでチェックできるのです。



正規表現と電話番号を照合

正規表現の書き方

では正規表現の具体的な記述方法を見ていきましょう。

文法 正規表現

```
/パターン/オプション
```

数字が4桁、先頭がアルファベットなどのルールの形式を**パターン**と呼びます。パターンとチェック対象の文字列を照合することで、任意の形式に沿っているかどうかを調べます。パターン内では「直前の文字が何文字分続くか」を表現する繰り返し文字を記述できます。また、オプションを付けてチェック方法を細かく指定することができます。

表7.4　パターンの表現方法一覧

文字	説明
[ ]	囲まれた中のいずれか一文字に一致する
-	[ ] の中に記述した場合、文字の範囲を表わす
^	文字列の先頭を意味する。[ ] の先頭に記述した場合は否定の意味になる
$	文字列の末尾を意味する
.	改行以外の一文字と一致する
¥n	改行と一致する
|	複数の文字列のうちいずれかと一致する
( )	パターンをグループ化する。グループ化するとグループ全体に対して繰り返し文字を適用できる

表7.5　繰り返し文字一覧

文字	説明
?	直前の文字 0 文字分か 1 文字分と一致する
*	直前の文字が 0 文字以上連続する箇所と一致する
+	直前の文字が 1 文字以上連続する箇所と一致する
{n}	直前の文字 n 文字分と一致する
{n,}	直前の文字が n 文字以上連続する箇所と一致する
{n,m}	直前の文字が n 文字以上、m 文字以下の範囲で連続する箇所と一致する

表7.6　オプション一覧

文字	説明
i	大文字と小文字を区別せずに照合する。
g	パターンと一致する部分文字列が複数ある場合、最初の 1 つだけでなくすべて検索する。
m	複数行に渡る文字列に対して照合を行なう場合に指定する。 ^ は各行の先頭に、$ は各行の末尾に一致するようになる

◇ パターンの記述例

パターンの記述例をいくつか紹介します。

パターン1 ▶ 文字列中にabcが含まれているかどうかをチェックする

```
/abc/
```

パターン2 ▶ 文字列中にaまたはbまたはcが含まれているかどうかをチェックする

```
/[abc]/
```

パターン3 ▶ 文字列中にa～z、A～Zまでのいずれかの文字が含まれているかどうかをチェックする

```
/[a-zA-Z]/
```

パターン4 ▶ 文字列中にアルファベットが含まれていないことをチェックする

```
/[^a-zA-Z]/
```

パターン5 ▶ 文字列中にabcまたはxyzが含まれているかどうかをチェックする

```
/abc|xyz/
```

これらを組み合わせて電話番号をチェックする正規表現を記述してみましょう。電話番号か調べるには、次のルールにのっとっているかどうかをチェックします。

- 利用できる文字は数字とハイフンのみ
- 先頭は0で始まる
- 市外局番は1～4桁
- 市内局番は1～4桁
- 加入者番号は4桁
- ハイフンの有無は問わない

これらをすべて正規表現で記述したものが以下です。

■電話番号チェックの正規表現

```
/^0[0-9]{1,4}-?[0-9]{1,4}-?[0-9]{4}$/
```

正規表現の利用

それでは、正規表現を利用してStringオブジェクトのmatchメソッドとreplaceメソッドを使ってみましょう。

まずは正規表現とmatchメソッドによる文字列の抽出例です。

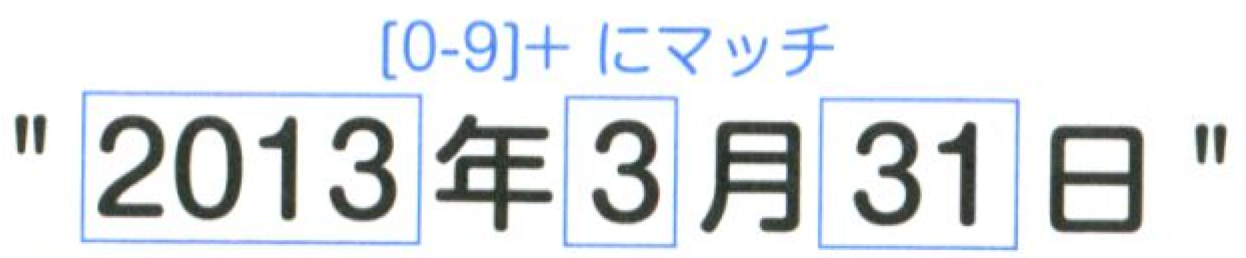
リスト7.14　matchメソッドの利用

```
var date = "2013年3月31日";
var result = date.match(/[0-9]+/g);
console.log(result);
```

■実行結果

```
["2013", "3", "31"]
```

リスト7.14では、日付を表わす文字列の中から「1文字以上連続した数字」というパターンに一致する部分を探しています。オプションとしてgが指定されているので、一致する部分が複数あればすべて取得します。結果、数字の部分だけが抜き出されて要素3つ分の配列となって返されます。



実行イメージ

続いて、正規表現とreplaceメソッドによる置換の例を見てみましょう。

リスト7.15 replaceメソッドの利用

```
var message = "This is a pen";
var result = message.replace(/is/g, "at");
console.log(result);
```

■実行結果

```
That at a pen
```

リスト7.15では、"This is a pen"という文字列の中から"is"という文字列に一致する部分を探し、"at"に置き換えています。オプションにgが付いているので、一致する部分すべてに対して置換が行なわれます。



実行イメージ

Point!



▶ Webサイトの入力チェックは正規表現を使って行なう。

この章の到達度チェック

Q1 空欄AとB を埋め、item2が持つcalcTaxメソッドを、item1のメソッドであるかのように呼び出すスクリプトを完成させてください。

```
var item1 = {
    price: 1000
};
var item2 = {
    price: 2000,
    calcTax: function() {
        console.log(this.price * 0.05);
    }
};
 [  A  ].calcTax.call( [  B  ]);
```

Q2 Q1のスクリプトが正しく実行されると、どのような結果が表示されますか？

Q3 以下のスクリプトを実行した結果、arrayとresultが同じ内容の配列を持つようにしたいと考えています。空欄を埋め、スクリプトを完成させてください。

```
var array = ["A", "B", "C"];
var str = array.join("/");
var result = str.[      ]("/");
```

Q4 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
var str = "money monday month common";
var words = str.match(/mon/g);
console.log(words.length);
```

Q5 以下のスクリプトを実行すると、どのような結果が表示されますか？

```
var str = "friday saturday sunday";
var rep = str.replace(/day$/g, "flower");
console.log(rep);
```

➲解答は巻末に掲載

第8章

イベント

この章で学ぶこと

- イベント駆動型モデル
- イベントハンドラの種類



なにかのきっかけで処理を行なう



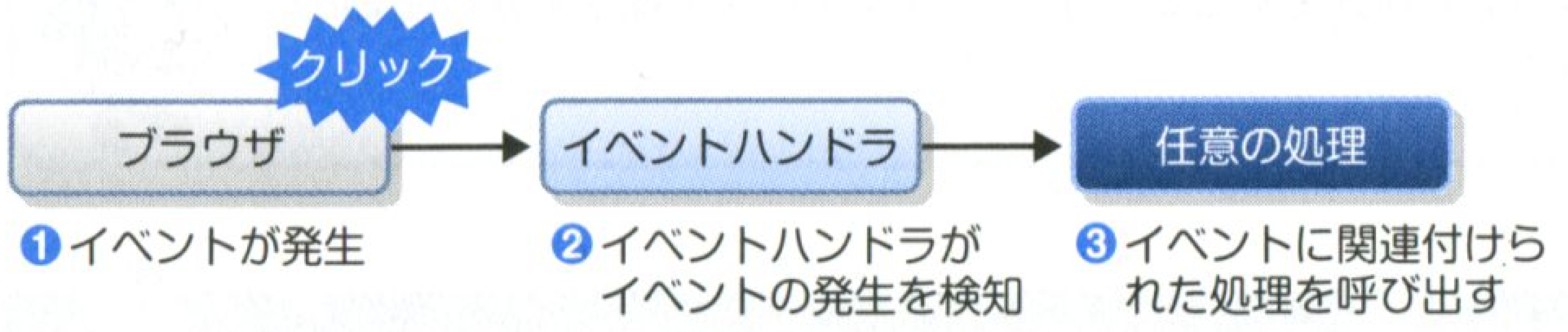
前章までは、ページを開くと即座にJavaScriptが実行されていました。本章では、ボタンがクリックされたときなど、なんらかのイベントが発生したタイミングでJavaScriptを実行する方法を学びます。

8 1 イベント駆動型モデル

イベントとは

ページが表示された、ボタンがクリックされたなど、Webページ上で発生した出来事を**イベント**と呼びます。また、イベントが発生したときに対応する処理を呼び出す仕組みを**イベント駆動型モデル**と呼びます。

JavaScriptは、イベントに応じた処理を実行する、イベント駆動型モデルのスクリプト言語です。それぞれのイベントと任意の処理をあらかじめ関連付けておくと、**イベントハンドラ**と呼ばれる機能がイベントの発生を検知し、対応する処理を呼び出します。



イベント駆動型モデル

主なイベント

表8.1は、よく使われるイベントの一覧です。イベントハンドラの名前は、「on」+イベント名となります。

表8.1 イベント一覧

種類	発生するタイミング	イベントハンドラ名
load	ページが表示されたとき	onload
click	ページ上の部品がクリックされたとき	onclick
mouseover	ページ上の部品にマウスカーソルが乗ったとき	onmouseover
mouseout	ページ上の部品からマウスカーソルが外れたとき	onmouseout
focus	ページ上の部品にフォーカスが当たったとき	onfocus
blur	ページ上の部品からフォーカスが外れたとき	onblur
change	セレクトボックスなどの選択肢が変更されたとき	onchange

Note 部品とは、第2章で触れたとおり、テキストボックスやボタン、セレクトボックス、リストなど、Webページ（フォーム）を形作るHTML要素を指します。

Point!

▶ イベントが発生すると、イベントハンドラが対応する処理を呼び出す。



■ Check Test 1

Q1 以下はイベントに関する説明です。空欄AとBを適切な語で埋めてください。

・Webページ上でボタンのクリックなどのイベントが起こると、[A]がイベントの発生を検知し、関連付けられた任意の処理を呼び出す。このような仕組みを[B]と呼ぶ。

➲解答は巻末に掲載

8 2 イベントの使い方

clickイベント

イベントが発生したタイミングで処理を行なうには、Webページとイベント（イベントハンドラ）を関連付ける必要があります。イベントハンドラは、HTMLタグの属性です。属性値の部分に、イベント発生時に実行する処理（JavaScript）を記述します。

文法

```
<タグ名 イベントハンドラ名="JavaScriptによる処理"></タグ名>
```

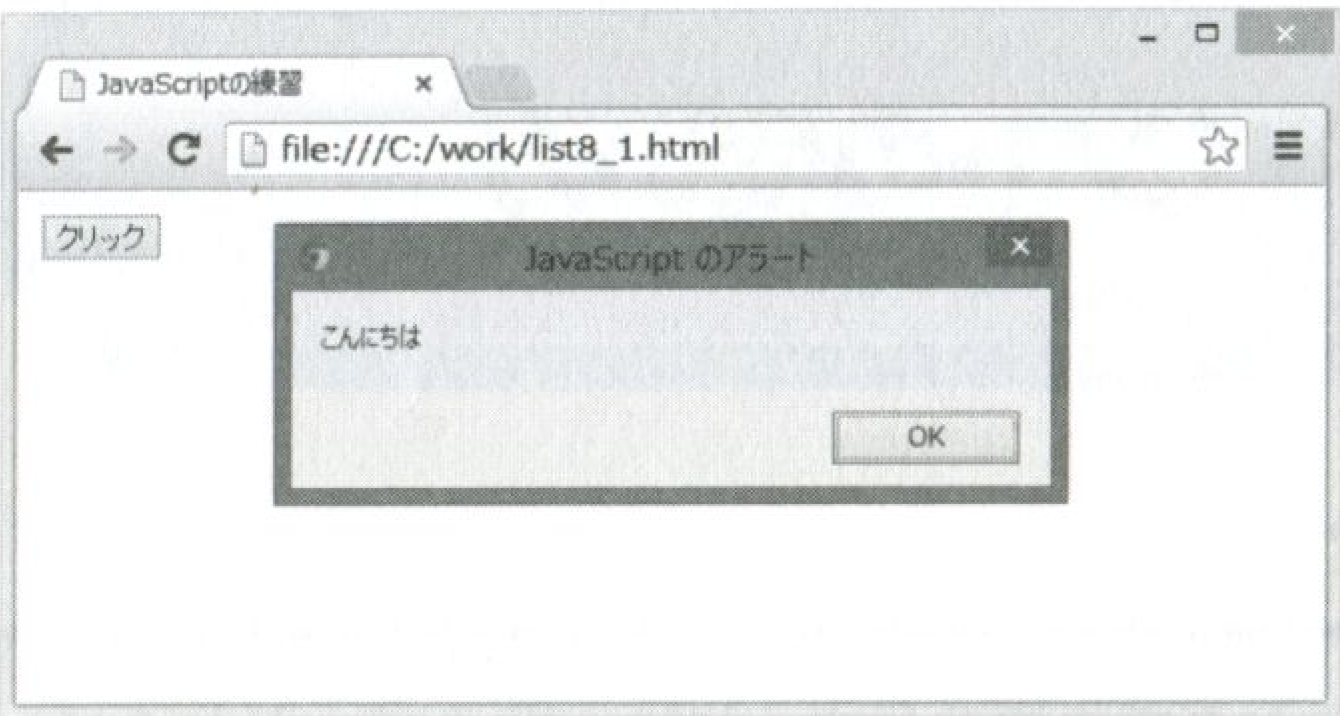
（イベントハンドラ名="JavaScriptによる処理"：HTMLタグの属性としてイベントハンドラを記述）

よく使うイベントの1つに、ボタンをクリックしたときに発生するclickイベントがあります。この利用例で記述方法をおぼえましょう。

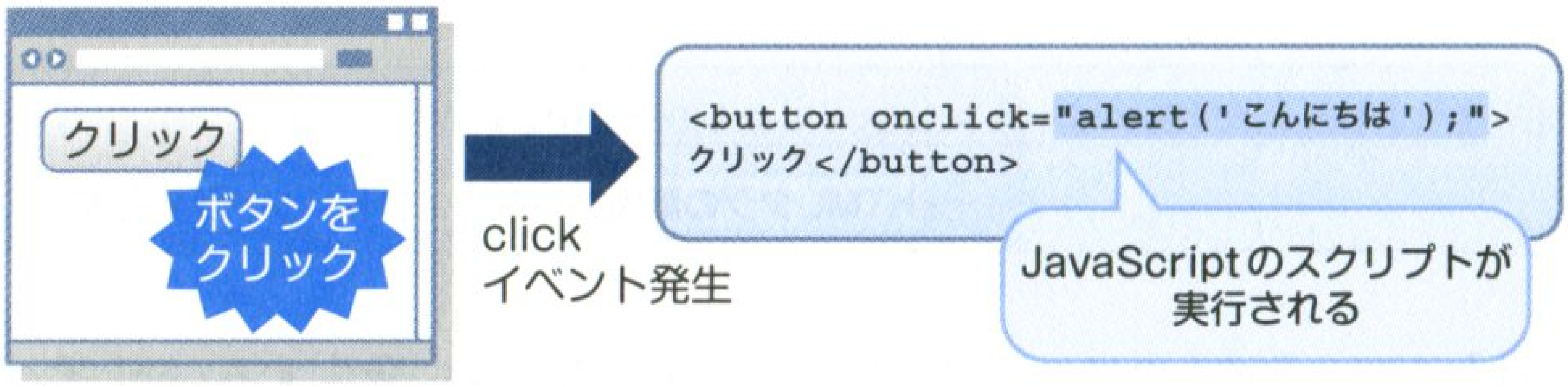
リスト8.1 clickイベントの利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
 </head>
 <body>
  <button onclick="alert('こんにちは');">クリック</button>
 </body>
</html>
```

■実行結果　ボタンクリック後



リスト8.1では、<button>タグにonclick属性を付与しています。このように処理を属性値として指定しておくと、ボタンがクリックされたタイミングで実行されます。



clickイベントの実行イメージ

注意しなければならないのは、

- ダブルクォートで囲まれた中（"～"）にはダブルクォートを記述しない
- シングルクォートで囲まれた中（'～'）の中にはシングルクォートを記述しない

ということです。もしリスト8.1の「こんにちは」という文字列をダブルクォートで囲んだ場合、「alert(」までがonclickの属性値となり、以降の命令は無効な文字列となってしまいます。

```
<button onclick="alert("こんにちは");">クリック</button>
```

ここまでが属性値（"alert("）　無効な文字列（こんにちは");"）

外側をダブルクォートにした場合は内側をシングルクォートにする、といったように、外側の囲み記号と内側の囲み記号は違うものを使うようにしましょう。

なお、この例は1行だけの短い処理なので命令文を属性値に直接記述しましたが、ほとんどの場合、複数行に渡る長い処理を実行することが多いはずです。そういった場合は、関数を呼び出す記述にします。次はリスト8.1の変更例です。

リスト8.1A イベント発生時に行なう処理をまとめた関数をイベントハンドラに指定

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function sayHello() {
    alert("こんにちは");
   }
  </script>
 </head>
 <body>
  <button onclick="sayHello();">クリック</button>
 </body>
</html>
```

クリック時に呼び出される

ここでは例としてclickイベントを取り上げましたが、他にもよく使うイベントがあります。以降では、Webアプリケーションで特に利用頻度の高いイベントについて見ていきましょう。

loadイベント

Webページが表示されたタイミングで実行されます。一般的に<body>要素の属性として指定します。

リスト8.2 loadイベントの利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
 </head>
 <body onload="console.log('こんにちは');">
 </body>
</html>
```

■実行結果

```
こんにちは
```

一見、前章までのようにイベントを設定せずに処理を記述したときと同じタイミングで動いているように見えますが、実際はわずかに異なります。<script>要素内や.jsファイル内に直接記述した処理は、<script>要素以降のHTMLタグが読み込まれる前に実行されます。それに対して、loadイベントに関連付けた処理はすべてのHTMLタグの読み込みが終了した後で実行されます。この違いは第10章のDOM（Document Object Model）を利用する際に重要なポイントとなります。

■〈script〉要素内に記述した場合の実行タイミング

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">                          ❶タグの読み込みを中断
  <title>JavaScriptの練習</title>
  <script>
   console.log("こんにちは");                      ❷実行
  </script>
 </head>
 <body onload="console.log('こんにちは');">       ❸タグの読み込みを再開
 </body>
</html>
```

■loadイベントの実行タイミング

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
 </head>
 <body onload="console.log('こんにちは');">
 </body>
</html>
```

❶タグをすべて読み込む

❷実行

mouseover ／ mouseoutイベント

ページ上の部品にマウスカーソルが乗ったときと、マウスカーソルが外れたときに実行されます。画像にマウスカーソルが乗ったら別の画像を表示するなどの用途で利用されます。

リスト8.3　mouseover ／ mouseoutイベントの利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
 </head>
 <body>
  <img src="images/frog1.jpg" onmouseover="console.log('カー ↵
ソルが乗っています');" onmouseout="console.log('カーソルが外れました');">
 </body>
</html>
```

■実行結果





focus／blurイベント

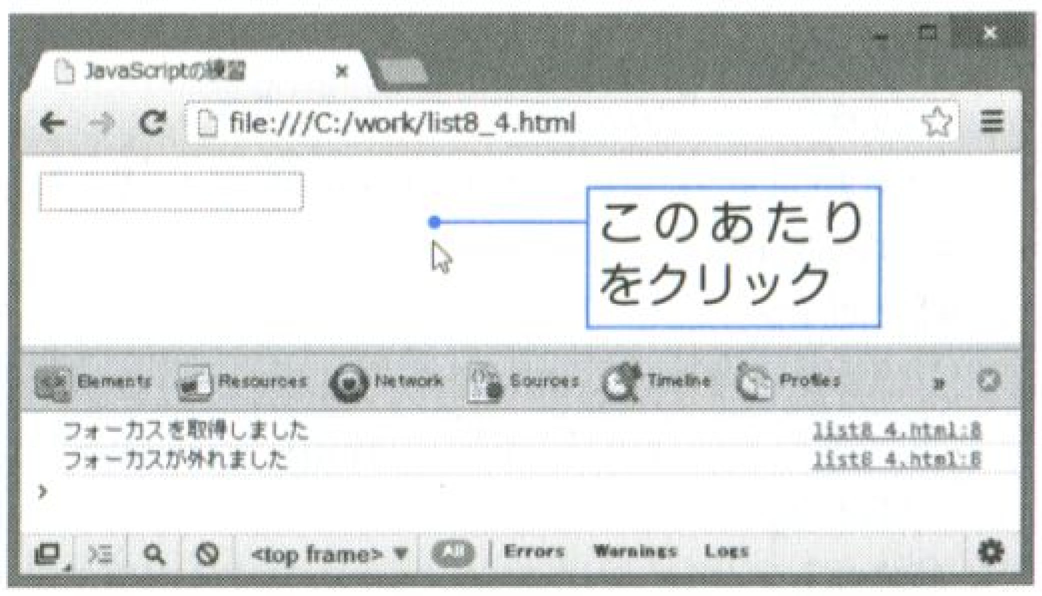
ページ上の部品にフォーカスが当たったときと、フォーカスが外れたときに実行されます。**フォーカス**とは、部品が選択された状態のことを指します。テキストボックスに入力カーソルが現われている、などがフォーカスを得ている状態です。

リスト8.4　focus／blurイベントの利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
 </head>
 <body>
  <input type="text" onfocus="console.log('フォーカスを取得 ↵
しました');" onblur="console.log('フォーカスが外れました');">
 </body>
</html>
```


■実行結果





onchangeイベント

セレクトボックスなどの選択項目が変更された場合に実行されます。たとえば、セレクトボックスの中から路線名が選択されたら、ページに駅名の一覧を表示する、といった処理を行なう場合などに利用します。

リスト8.5 onchangeイベントの利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title> JavaScriptの練習</title>
 </head>
 <body>
  <select onchange="console.log('項目が変更されました');">
   <option>りんご</option>
   <option>いちご</option>
   <option>みかん</option>
  </select>
 </body>
</html>
```

■実行結果



Point!

▶ イベントハンドラは要素の属性値として記述する。



■ Check Test 2

Q1 onloadイベントハンドラは、一般的にどの要素に対して関連付けますか？

Q2 部品からフォーカスが外れたタイミングで発生するイベントはなんですか？

解答は巻末に掲載

8-3 イベント発生元の特定

thisキーワードの利用

1つの関数を複数のイベントから呼び出すとき、呼び出された関数内でどの部品から発生したイベントなのかを知りたい場合があります。そのようなときには、thisキーワードを利用します。イベントハンドラに関連付けた処理の中でthisキーワードを記述した場合、thisはイベントハンドラ属性が付けられたHTML要素自身を表わします。

リスト8.6 イベントの発生元を調べる

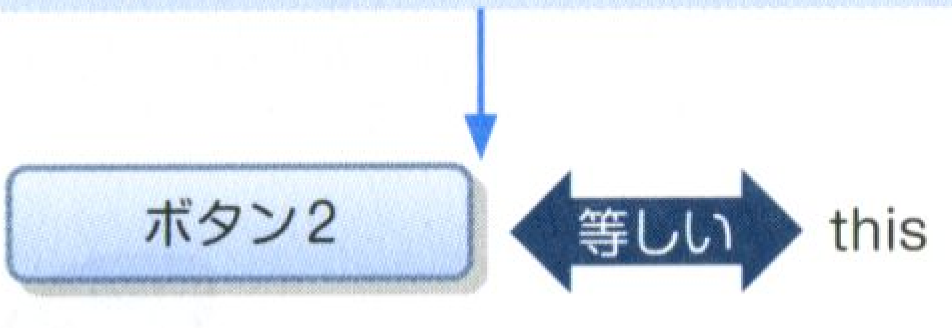
```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function check(e) {
    console.log(e.value + "がクリックされました");
   }
  </script>
 </head>
 <body>
  <input type="button" value="ボタン1" onclick="check(this);">
  <input type="button" value="ボタン2" onclick="check(this);">
  <input type="button" value="ボタン3" onclick="check(this);">
 </body>
</html>
```

■実行結果



リスト8.6では、まず各部品のonclick属性でcheck関数を呼び出し、引数としてthisを渡しています。このとき、thisはイベント発生元の要素、仮に［ボタン2］がクリックされた場合は以下の<input>要素そのものを表わしています。

```
<input type="button" value="ボタン2" onclick="check(this);">
```



check関数は、thisを引数として受け取り、変数eに代入しています。変数eの後ろに「.属性名」と続けることでイベント発生元要素の各属性値を得ることができます。

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function check(e) {
       console.log(e.value + "がクリックされました");
   }
  </script>
 </head>
 <body>
  <input type="button" value="ボタン1" onclick="check(this);">
  <input type="button" value="ボタン2" onclick="check(this);">
  <input type="button" value="ボタン3" onclick="check(this);">
 </body>
</html>
```

自分自身の情報を渡す

Point!

- イベント処理におけるthisキーワードはイベントの発生元に関する情報を持つ。



この章の到達度チェック

Q1 以下のタグでは、ページが表示されたタイミングでコンソールにメッセージを表示する処理を設定しています。空欄を埋め、処理を完成させてください。

```
<body [      ]="console.log('Welcome');">
```

Q2 以下のタグでは、テキストボックスからフォーカスが外れたタイミングでコンソールにメッセージを表示する処理を設定しています。空欄を埋め、処理を完成させてください。

```
<input type="text" [      ]="console.log('フォーカスが外れました');">
```

Q3 以下のタグでは、セレクトボックスの選択項目が変更されたタイミングでコンソールにメッセージを表示する処理を設定しています。空欄を埋め、処理を完成させてください。

```
<select [      ]="console.log('Hello');">
```

Q4 以下のタグでは、ボタンがクリックされたタイミングでfunc関数を呼び出し、引数として自分自身を渡す処理を設定しています。空欄を埋め、処理を完成させてください。

```
<button onclick="func([      ]);">ボタン</button>
```

Q5 変数elmにはイベント発生元の要素が代入されています。要素のvalue属性の値を得るには、空欄になにが入りますか？

```
elm.[      ]
```

➲解答は巻末に掲載

8

第9章

ブラウザオブジェクト

この章で学ぶこと

- ブラウザオブジェクトの種類
- タイマー機能
- ページ間の移動
- フォーム要素の取得



● ブラウザオブジェクトで、Webページを操作できる ●



ブラウザやWebページに関する情報を持つオブジェクトを、ブラウザオブジェクトと呼びます。本章ではブラウザオブジェクトの種類とその機能について学びます。

9-1 ブラウザオブジェクト

ブラウザオブジェクトとは

ブラウザ上の情報は、すべてオブジェクトとして扱うことができます。これらのオブジェクトを総称して**ブラウザオブジェクト**と呼びます。各ブラウザオブジェクトは、ブラウザ上でさまざまな操作を行なうための機能を提供します。表9.1は、よく使われるブラウザオブジェクトです。

表9.1　主なブラウザオブジェクト一覧

オブジェクト名	説明
Window	すべてのブラウザオブジェクトの親となるオブジェクト（ブラウザのウィンドウ全体の情報を持つオブジェクト）
Location	Web ページの URL を情報として持つオブジェクト
History	Web ページの閲覧履歴を情報として持つオブジェクト
Document	Web ページ上の部品の情報を持つオブジェクト（Web ページの情報を持つオブジェクト）

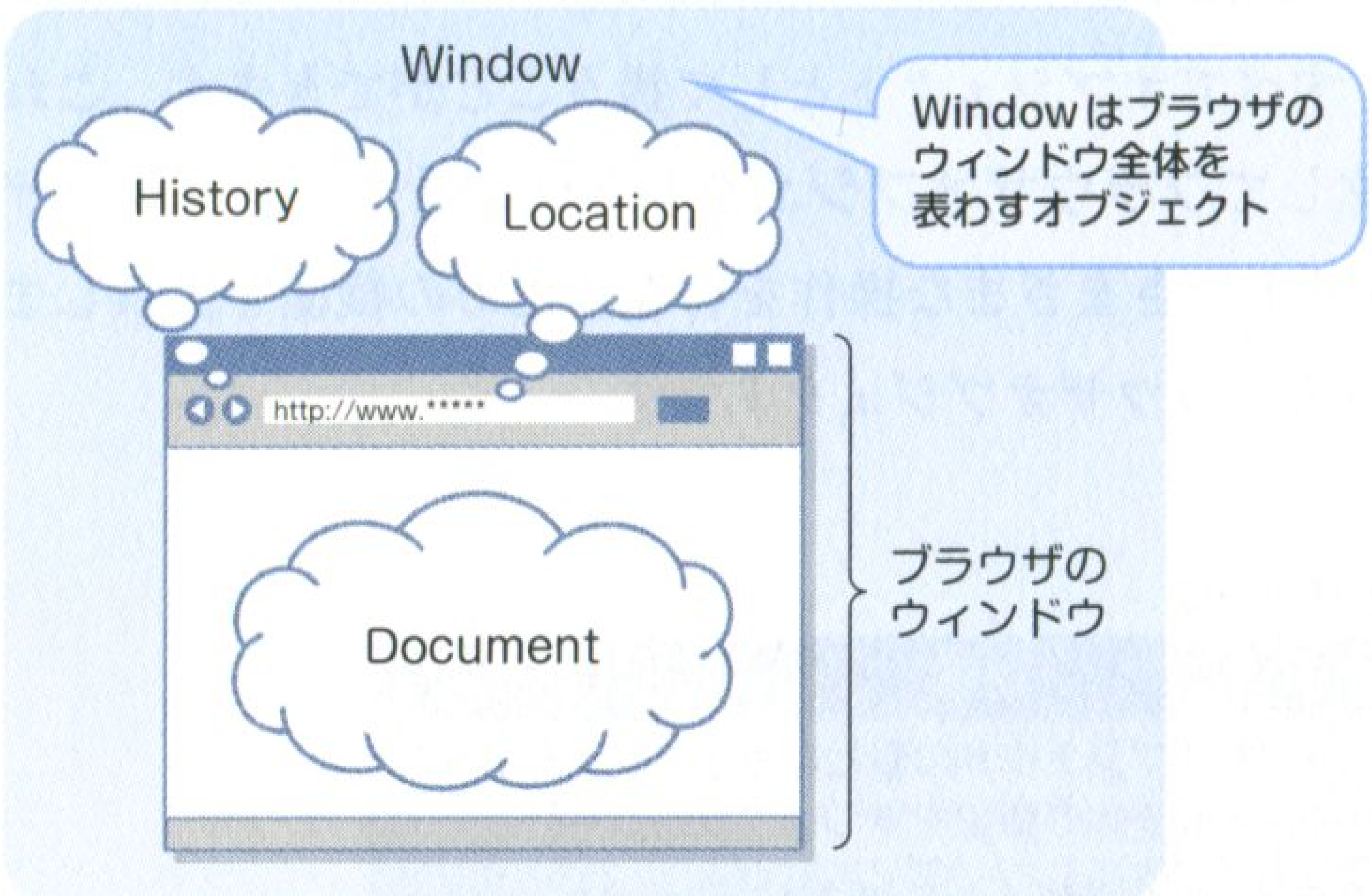
Note
ブラウザオブジェクトは、ブラウザに組み込まれているオブジェクトです。ブラウザの種類によってはブラウザオブジェクト（以降で紹介するプロパティやメソッド）の挙動が若干異なる場合もありますが、ここで紹介する主要なものについてはほとんど動作に違いはありません。

ブラウザオブジェクトも多くのメソッドやプロパティを持っています。ブラウザオブジェクトのメソッドやプロパティを利用する場合も、先頭にオブジェクト名を指定しますが、この際、オブジェクト名の頭文字は小文字にします。

Window ➡window
Location ➡location

History ➡history
Document➡document

Windowオブジェクトは、ブラウザのウィンドウ全体を表わすオブジェクトです。すべてのブラウザオブジェクトはWindowオブジェクトに内包されています。



すべてのブラウザオブジェクトはWindowオブジェクトに含まれる

そのため、ブラウザオブジェクトを利用するときは、Windowオブジェクトを先頭に記述します。たとえば、Locationオブジェクトのhrefプロパティを利用する場合、次のように記述します。

```
window.location.href
```

ただし、先頭のWindowオブジェクト、つまり「window.」は省略しても良いことになっています。よって、次のように記述しても同じ意味であり、通例ではこのように記述します。以降、本書ではすべての命令から「window.」を省いた状態で記載します。

```
location.href
```

次節から、ブラウザオブジェクトの中でも特によく利用されるWindow、

Location、History、Documentをもう少し詳しく見ていきましょう。

Point!

▶ ブラウザオブジェクトのメソッドやプロパティを呼び出すとき、オブジェクト名の頭文字は小文字にする。

▶ 先頭のwindow.は省略できる。



■ Check Test 1

Q1 以下はWindowオブジェクトに関する説明です。空欄AとBを適切な語で埋めてください。

・すべてのブラウザオブジェクトの［ A ］となるオブジェクトで、［ B ］に関する情報を持つ。

➲解答は巻末に掲載

9-2 Windowオブジェクト

ブラウザのウィンドウ全体の情報

Windowオブジェクトは、すべてのブラウザオブジェクトの親となるオブジェクトで、ブラウザのウィンドウ全体の情報を保持します。ウィンドウ全般に関する操作を行なうことができます。前章までに使ってきたalert命令も、Windowオブジェクトのメソッドの1つです。

表9.2 Windowオブジェクトの主なメソッド一覧

メソッド名	概要
alert	警告用のダイアログボックスを表示する
confirm	確認用のダイアログボックスを表示する
open	サブウィンドウを開く
close	サブウィンドウを閉じる
setInterval	タイマーを設定する
clearInterval	タイマーを解除する

これらの使い方を見ていきましょう。

confirmメソッド

ダイアログボックスを表示します。confirmメソッドはOKとキャンセルの2つのボタンを表示し、どちらのボタンがクリックされたかを示す論理値を戻り値として返します。

文法 confirmメソッド

```
confirm("表示するメッセージ");
```

戻り値 true…OK／false…キャンセル

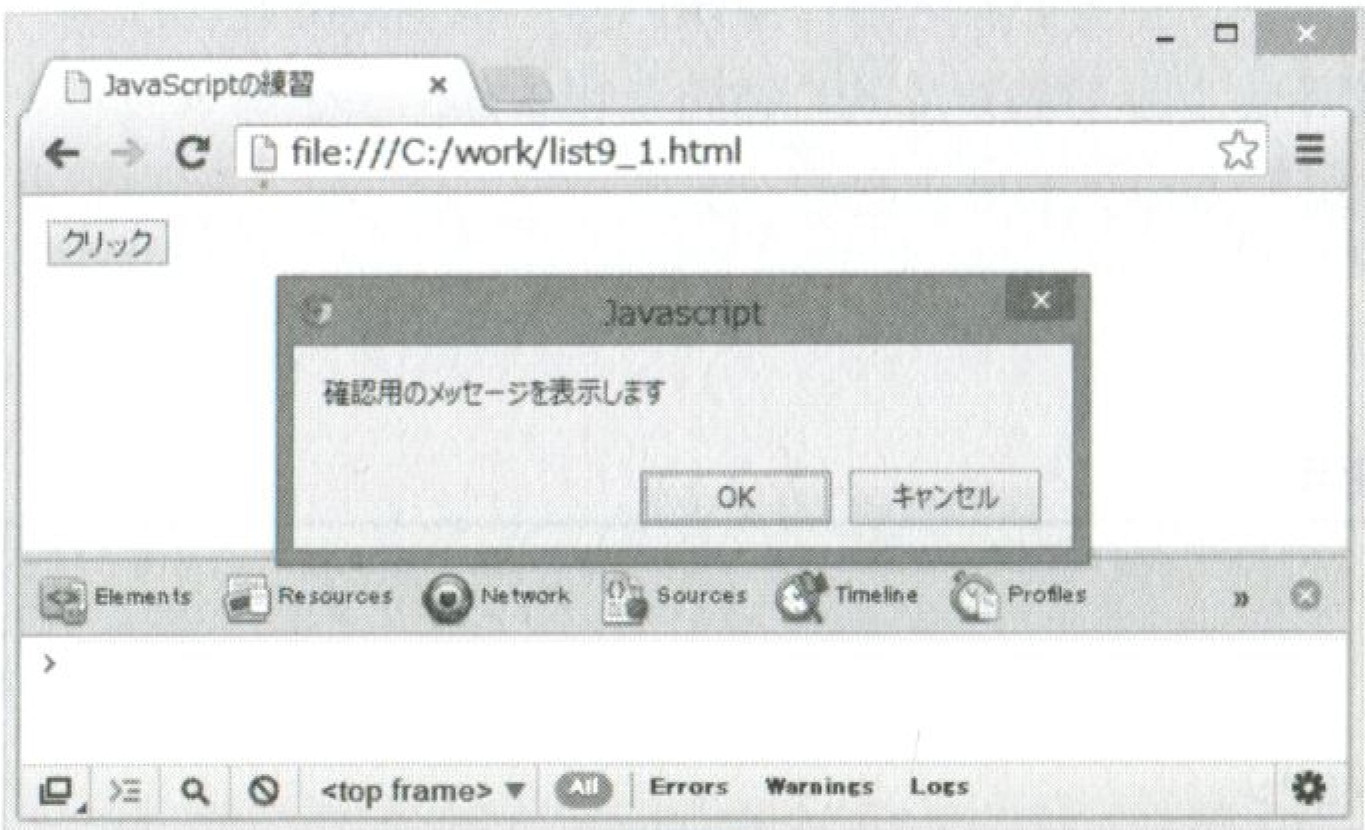
Note 前節で説明したとおり、オブジェクト名の「window.」は省略するのが通例です。以降で紹介するメソッドも基本的に省略しますが、closeメソッドだけは特別に省略しません。これはのちほど説明します。

リスト9.1 確認用ダイアログの表示

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function check() {
       var result = confirm("確認用のメッセージを表示します");
       if (result) {
           console.log("OK");
       } else {
           console.log("キャンセル");
       }
   }
  </script>
 </head>
 <body>
  <button onclick="check();">クリック</button>
 </body>
</html>
```

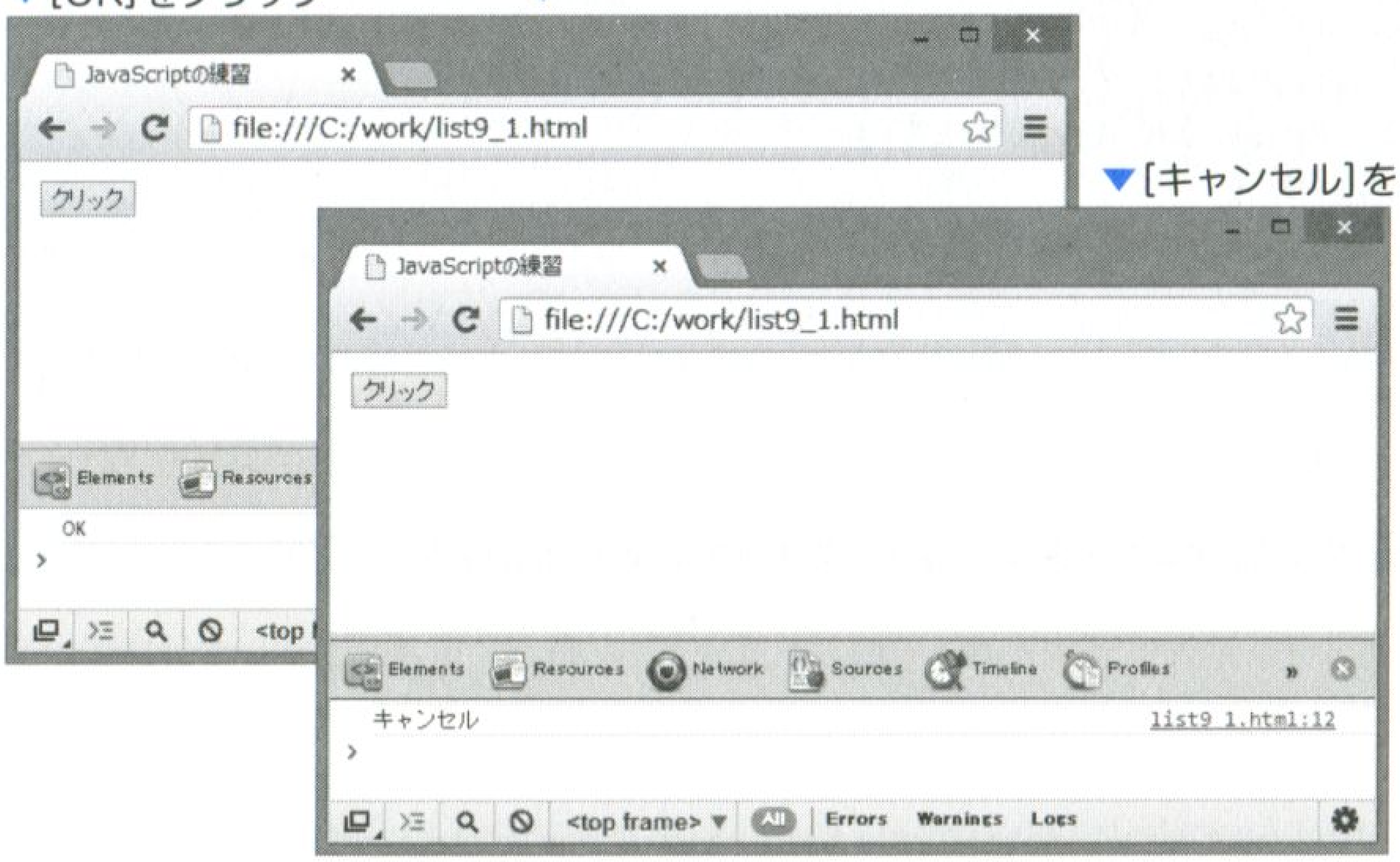
9

■実行結果



▼[OK]をクリック

▼[キャンセル]をクリック



openメソッドとcloseメソッド

openメソッドでブラウザの新規ウィンドウまたは新規タブを開き、closeメソッドで閉じます。ウィンドウとタブのどちらが表示されるかはブラウザの設定に依存します。

文法 openメソッド

```
open("URL", "ウィンドウ名", "オプション1=値,オプション2=値・・・");
```

第2引数以降は省略可能

openメソッドは、新規ウィンドウ／タブを開き、指定されたURLに移動します。第2引数以降は省略可能です。第3引数には、次のオプションを複数指定できます。ただし、新規タブとして表示された場合、オプションの設定は無視されます。

openメソッドのオプション一覧

オプション	値	説明
width	ピクセル値	ウィンドウの幅
height	ピクセル値	ウィンドウの高さ
top	ピクセル値	デスクトップ画面の上端からウィンドウまでの距離
left	ピクセル値	デスクトップ画面の左端からウィンドウまでの距離

現在のウィンドウ／タブを閉じるには、closeメソッドを使います。

文法 closeメソッド

```
window.close();
```

閉じることができるのは、openメソッドによって開かれたウィンドウ／タブのみです。なお、Documentオブジェクトにもcloseメソッドが存在するため、メソッド名のみを指定して呼び出すとうまく動作しません。Windowオブジェクトのcloseメソッドであることを明示するために、必ず「window.」は省略せずに記述してください。

リスト9.2は、list9_2A.htmlからlist9_2B.htmlを新規ウィンドウで開くスクリプトです。

リスト9.2 サブウィンドウの表示

HTML1 list9_2A.html

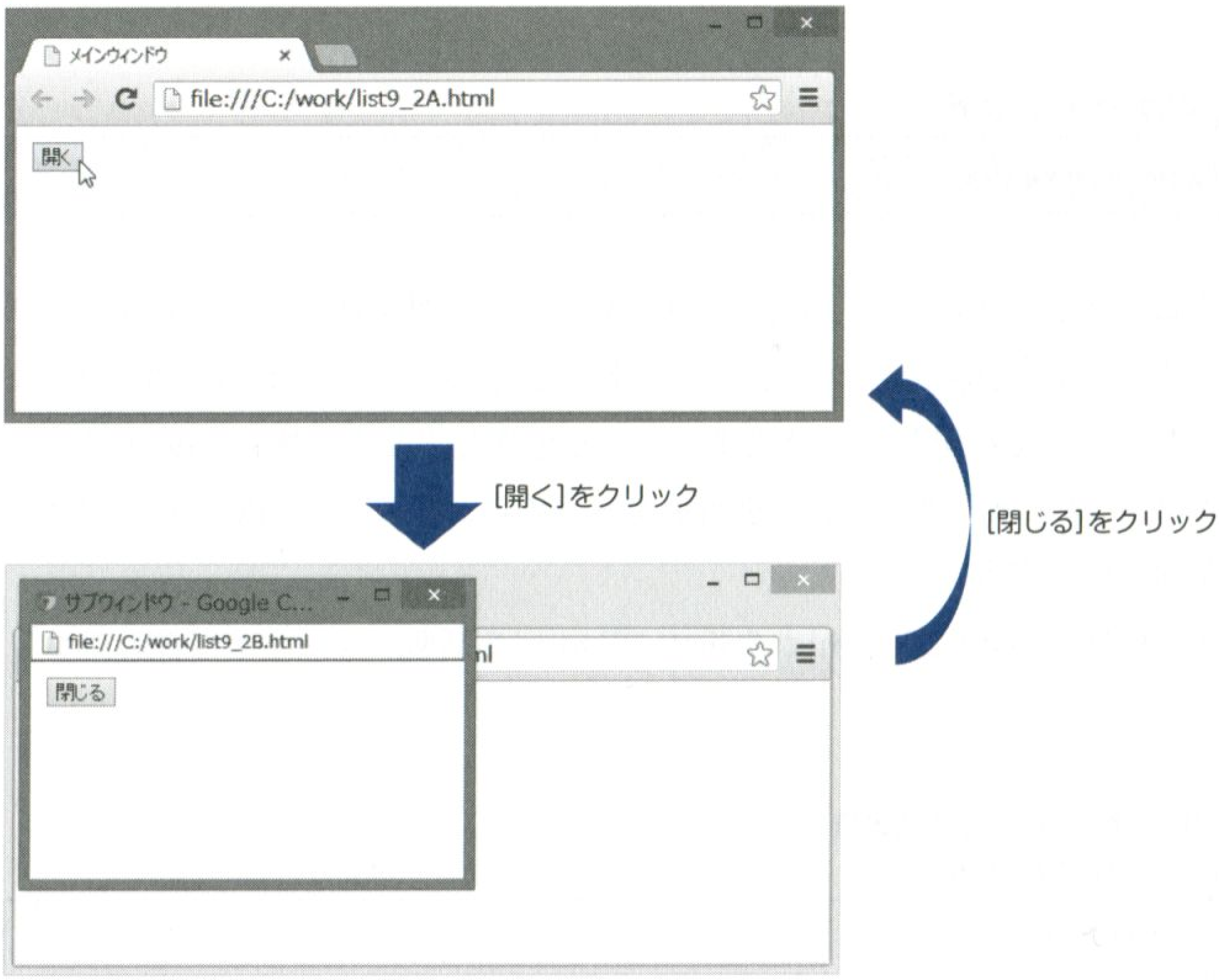
```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>メインウィンドウ</title>
 </head>
 <body>
  <button onclick="open('list9_2B.html', null, 'width= ⏎
```

```
300,height=150');">開く</button>
 </body>
</html>
```

HTML2 list9_2B.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>サブウィンドウ</title>
 </head>
 <body>
  <button onclick="window.close();">閉じる</button>
 </body>
</html>
```

■実行結果



setIntervalメソッドとclearIntervalメソッド

setIntervalメソッドは、一定の間隔ごとに処理を呼び出す、**タイマー**機能を開始します。一定間隔で表示画像を切り替えるなどの用途で使われます。タイマーを停止するには、clearIntervalメソッドを使います。

文法 setIntervalメソッド

```
setInterval(関数名または関数式, 時間間隔);
```
戻り値 タイマーID

時間間隔はミリ秒（1/1000秒）単位で指定します。時間間隔の精度はブラウザによって異なりますが、1秒以上大きくずれるようなことはまずありません。

タイマーが開始されると、指定された処理が一定間隔で実行され続けます。このメソッドの戻り値は、タイマーに割り当てられたIDです。IDは、タイマーを停止する際に必要となります。

タイマーを指定するには、clearIntervalメソッドを使います。このメソッドは、指定されたIDのタイマーを停止します。

文法 clearIntervalメソッド

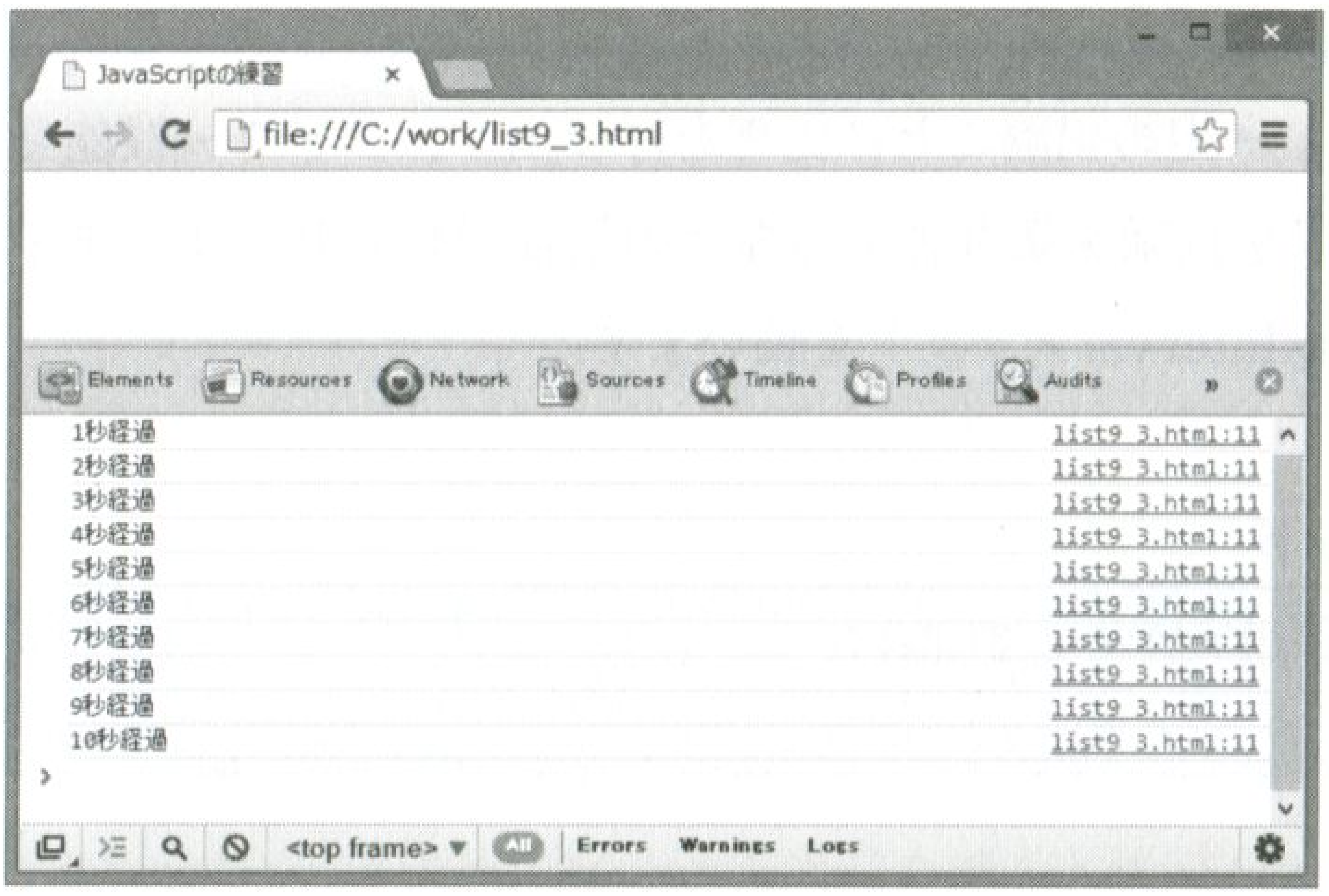
```
clearInterval(タイマーID);
```

タイマーの利用例を見てみましょう。

リスト9.3 タイマー機能の利用（JavaScript）

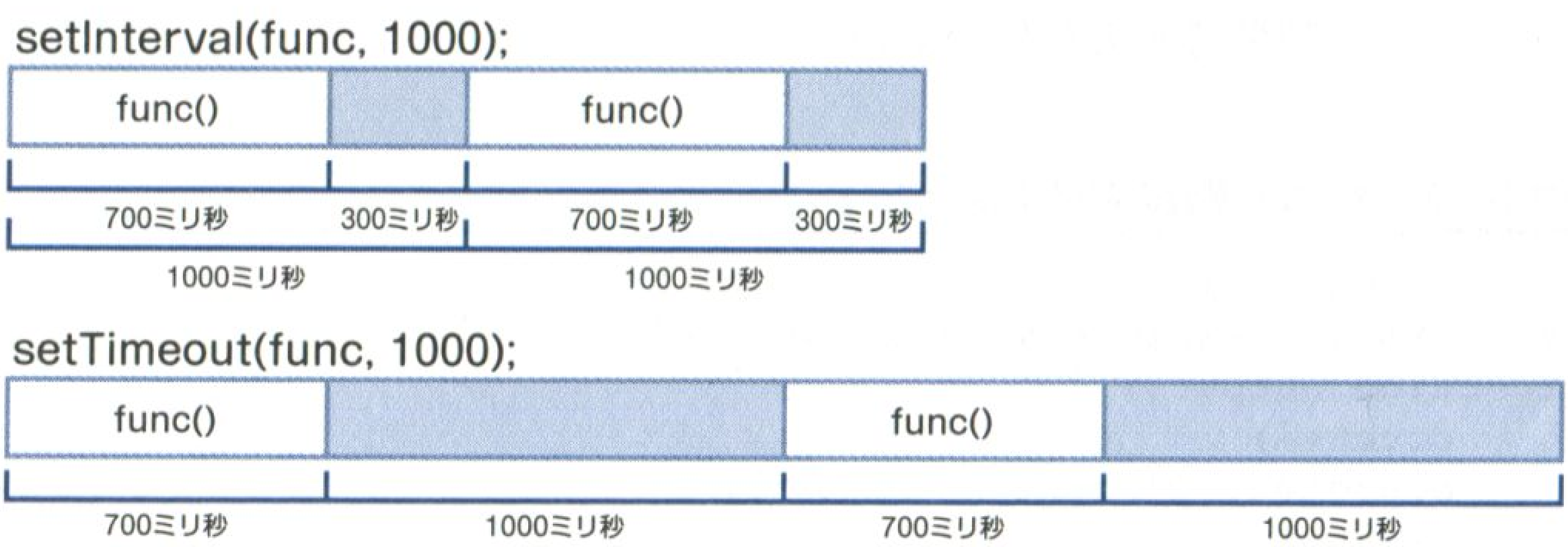
```
var count = 0;
var timer = setInterval(output, 1000);
function output() {
    count++;
    console.log(count + "秒経過");
    if(count >= 10) {
        clearInterval(timer);
    }
}
```

■実行結果



リスト9.3では、1000ミリ秒（1秒）ごとにoutput関数を呼び出しています。output関数は経過時間をコンソールに表示します。output関数が10回呼び出された時点で、タイマー機能を停止しています。

また、setInterval／clearIntervalによく似たメソッドに、setTimeout／clearTimeoutがあります。引数などの記述方法はどちらも同じですが、setInterval／clearIntervalが処理を開始する間隔を指定するのに対し、setTimeout／clearTimeoutは処理が終了してから何秒後に次の処理を開始するのかを指定します。第14章で学ぶアニメーションのような、完了までに時間がかかる処理を実行する場合はsetTimeout／clearTimeoutを利用しましょう。



setIntervalとsetTimeoutの違い

Point!

▶ Windowオブジェクトはすべてのブラウザオブジェクトの親にあたるオブジェクトで、ダイアログボックスの表示やタイマー処理を行なうことができる。



■ Check Test 2

Q1 openメソッドで開いたウィンドウまたはタブを閉じるには、どのような命令を記述しますか？

Q2 setIntervalメソッドとsetTimeoutメソッドでは、どちらのほうが処理の実行間隔が長くなりますか？

➲解答は巻末に掲載

9-3 LocationオブジェクトとHistoryオブジェクト

WebページのURL情報（Location）

Locationオブジェクトは、WebページのURLを情報として持つオブジェクトです。hrefプロパティの値を変更することで、別のWebページに移動できます。

文法 現在のURLを書き換える

```
location.href = 移動先URL;
```

■記述例

```
location.href = "http://www.google.co.jp/";
```

9

Webページの履歴情報（History）

Historyオブジェクトは、Webページの閲覧履歴を情報として持つオブジェクトです。backメソッドで1つ前ページに戻ることができます。ブラウザの戻るボタンをクリックしたときと同じ挙動です。

文法 前のページに戻る

```
history.back();
```

リスト9.4は、location.hrefとhistory.back()を利用して2つのページ間を移動するスクリプトです。

リスト9.4 ページ間の移動

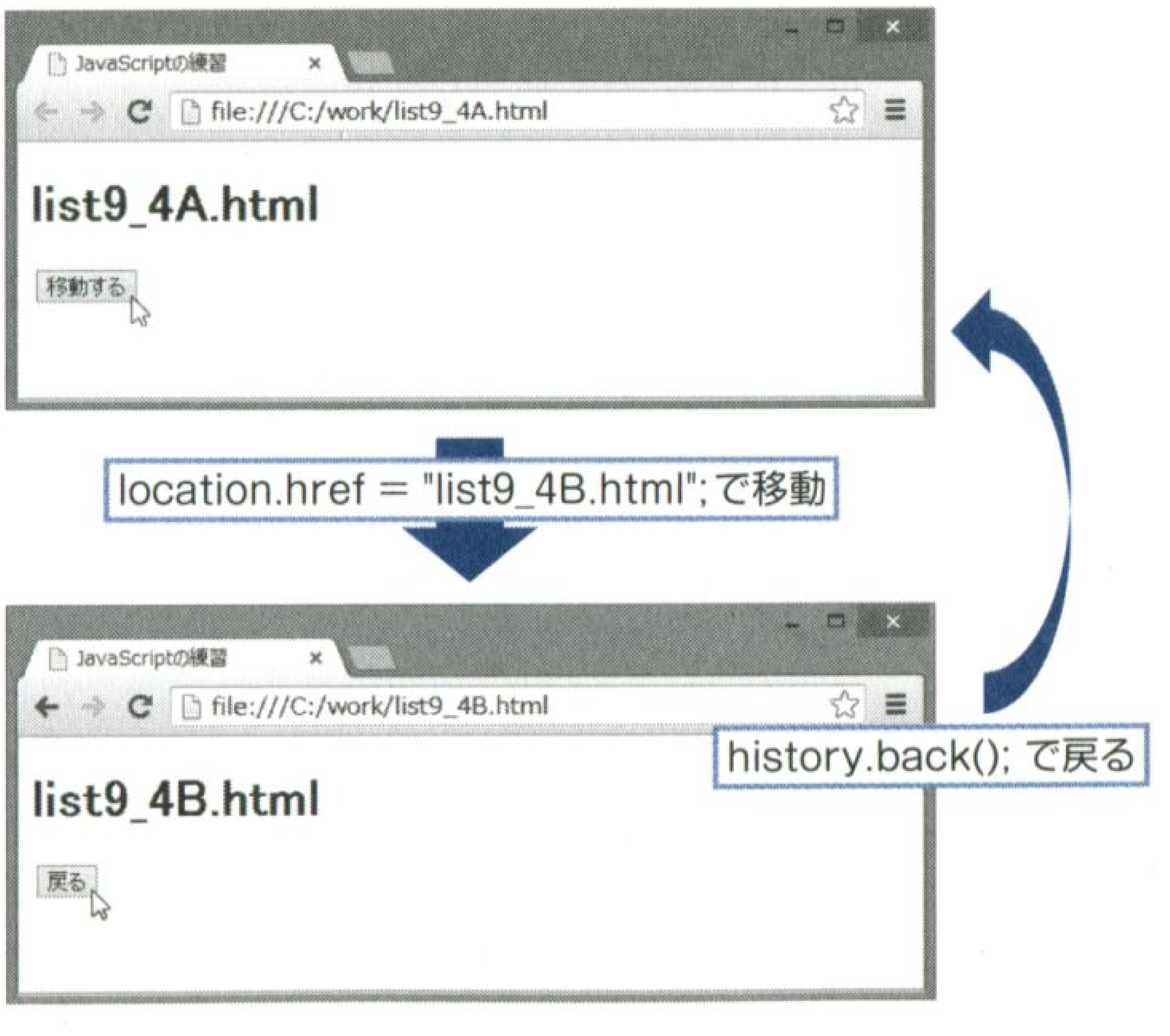
HTML1 list9_4A.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
 </head>
 <body>
  <h1>list9_4A.html</h1>
  <button onclick="location.href='list9_4B.html';">
移動する</button>
 </body>
</html>
```

HTML2 list9_4B.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
 </head>
 <body>
  <h1>list9_4B.html</h1>
  <button onclick="history.back();">戻る</button>
 </body>
</html>
```

■実行結果



Point!



▶ Locationオブジェクトや Historyオブジェクトで、ページ間の移動ができる。

■ Check Test 3

Q1 現在のURLをダイアログボックスで表示するには、どのような命令を記述しますか？

Q2 ブラウザの戻るボタンをクリックしたときの動きをJavaScriptで実現するには、どのような命令を記述しますか？

➲解答は巻末に掲載

9.4 Documentオブジェクト

Webページの表示情報

Documentオブジェクトは、Webページ上にあるすべての部品の情報を持つオブジェクトです。Documentオブジェクトを介して、個々のHTML要素にアクセスできます。「アクセス」とは、データの読み出し（取得）や書き出し（変更）を意味します。

HTML要素へアクセスするには、次章で学ぶgetElementByIdメソッドを利用する方法が一般的です。しかし、例外的にフォーム要素に対してのみ、name属性を指定してのアクセスが可能となっています。

文法 フォーム要素の取得

```
document.フォーム名.フォーム内要素名
```

■記述例

```
<body>
 <form name="frm">
  <input type="text" name="message">
 </form>
</body>
```

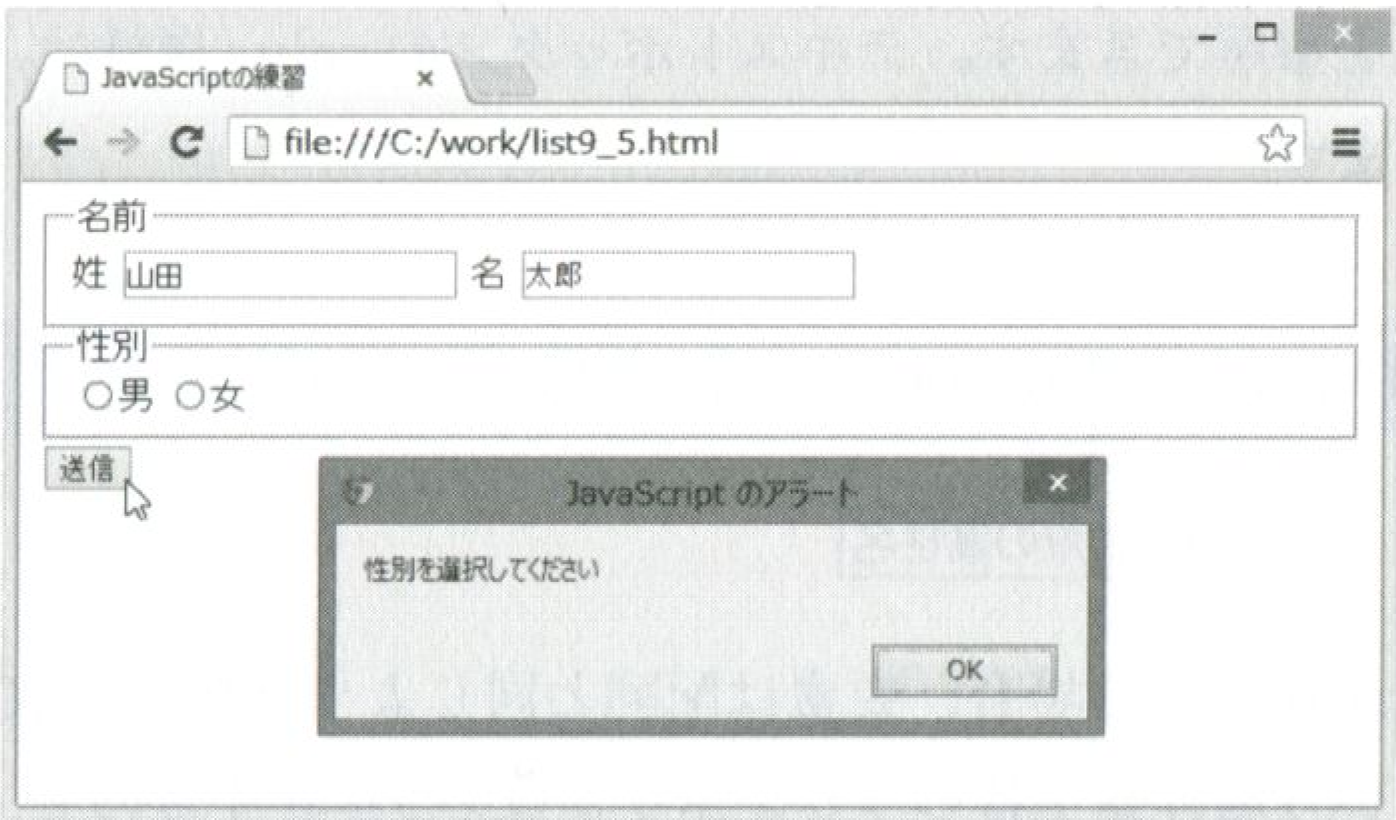
document.frm.messageでアクセスできる

リスト9.5は、送信ボタンをクリックするとフォームの入力チェックを行なうスクリプトです。

リスト9.5 フォームの入力チェック

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function inputCheck(){
       if(document.frm1.lastname.value == "") {  ❶
           alert("姓を入力してください");
       } else if (document.frm1.firstname.value == "") {
           alert("名を入力してください");
       } else if (!document.frm1.gender[0].checked &&  ❷
           !document.frm1.gender[1].checked) {
           alert("性別を選択してください");  ❸
       } else {
           document.frm1.submit();  ❹
       }
   }
  </script>
 </head>
 <body>
  <form name="frm1" action="xxx.html">
   <fieldset>
    <legend>名前</legend>
    <label>姓
     <input type="text" name="lastname">
    </label>
    <label>名
     <input type="text" name="firstname">
    </label>
   </fieldset>
   <fieldset>
    <legend>性別</legend>
    <label>
     <input type="radio" name="gender" value="m">男
    </label>
    <label>
     <input type="radio" name="gender" value="f">女
    </label>
   </fieldset>
   <input type="button" onclick="inputCheck();" value="送信">  ❺
  </form>
 </body>
</html>
```

■実行結果



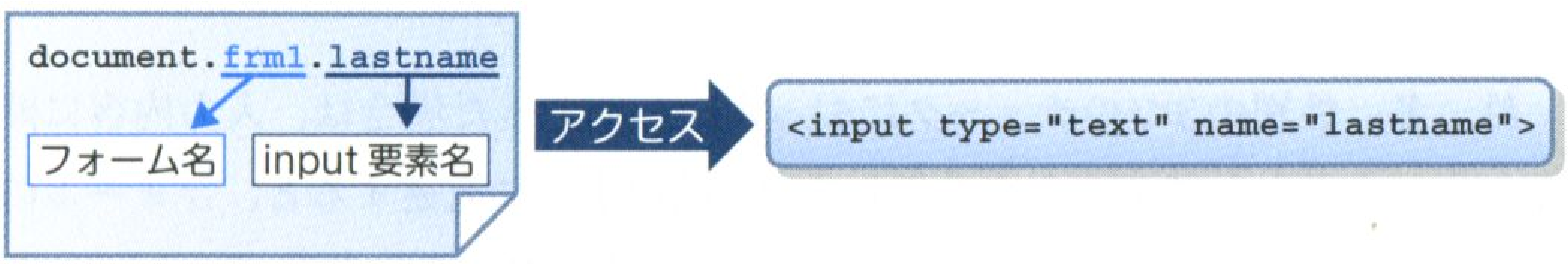
フォームの送信ボタンがクリックされたタイミングで、入力チェックを行なうinputCheck関数を呼び出しています。入力チェックを行なってNGだった場合は、データの送信をキャンセルします。そのため、リスト9.5の❺のように、ボタンの種類は送信ボタン（type="submit"）ではなく汎用ボタン（type="button"）にしています。

```
<input type="button" onclick="inputCheck();" value="送信">
```

inputCheck関数では、まず❶で「lastname」というname属性値を持つ要素にアクセスしています。

```
if(document.frm1.lastname.value == "") {
```

「フォーム名.要素名」という書式で、特定の要素にアクセスできます。



特定の要素にアクセスできたら、さらにドット（.）を挟んで要素の属性名をつなげることで、属性値を参照できます。テキストボックスのvalue属性は、入力された文字列を表わします。

```
if(document.frm1.lastname.value == "") {
```

<input>要素の属性名

name属性値が同じ要素が複数ある場合、要素は配列と同じような形式で取得されます。「gender」という属性値を持つラジオボタンは2つ存在するので、「document.frm1.gender」はインデックス0番目と1番目の2つの要素を持っていることになります。

```
document.frm1.gender[0]
  → <input type="radio" name="gender" value="m">
document.frm1.gender[1]
  → <input type="radio" name="gender" value="f">
```


❷では、オプションボタンの選択状態をチェックしています。オプションボタンとチェックボックスは、要素が選択された状態のとき、checked属性の値がtrueになります。2つのオプションボタンのchecked属性の状態がどちらもfalseであった場合は選択が行なわれていないことになるため、エラーメッセージを表示しています。

```
} else if (!document.frm1.gender[0].checked &&
    !document.frm1.gender[1].checked) {
```

姓、名、性別の3つのチェックに引っかからなかった場合は、入力内容に問題がないものとしてデータを送信します。❹のように記述すると、フォームの送信（submit）ボタンをクリックしたときと同様の働きをします。

```
document.frm1.submit();
```

また、要素の属性値は参照だけでなく変更も可能です。性別が未選択の状態で送信ボタンがクリックされた場合、男性を既定値として送信を実行するには、❸を次のように変更します。

```
alert("性別を選択してください");
```

↓変更

```
document.frm1.gender[0].checked = true;   // 男性を選択状態に変更
document.frm1.submit();                   // データ送信
```

次章では、DOMメソッドを利用した、より簡潔な要素へのアクセス方法について学びます。

Point!

▶ Documentオブジェクトを使ってHTML要素にアクセスすることができる。



この章の到達度チェック

Q1 以下のスクリプトを実行して、確認ダイアログで［OK］をクリックした場合、なにが起こりますか？

```
if( !confirm("よろしいですか？") ) {
    location.href = "http://www.google.co.jp/";
} else {
    history.back();
}
```

Q2 Q1のスクリプトを実行して、確認ダイアログで［キャンセル］をクリックした場合は何が起こりますか？

Q3 以下のスクリプトを実行すると、func関数は何秒おきに実行されますか？

```
var tid = setInterval(func, 5000);
```

Q4 Q3のタイマー処理を停止するには、どのような命令文を記述すれば良いですか？

Q5 以下の<input>要素のvalue属性に"ABC"をセットするには、どのような命令文を記述すれば良いですか？

```
<form name="fr">
 <input type="text" name="tx">
</form>
```

➲解答は巻末に掲載

第10章

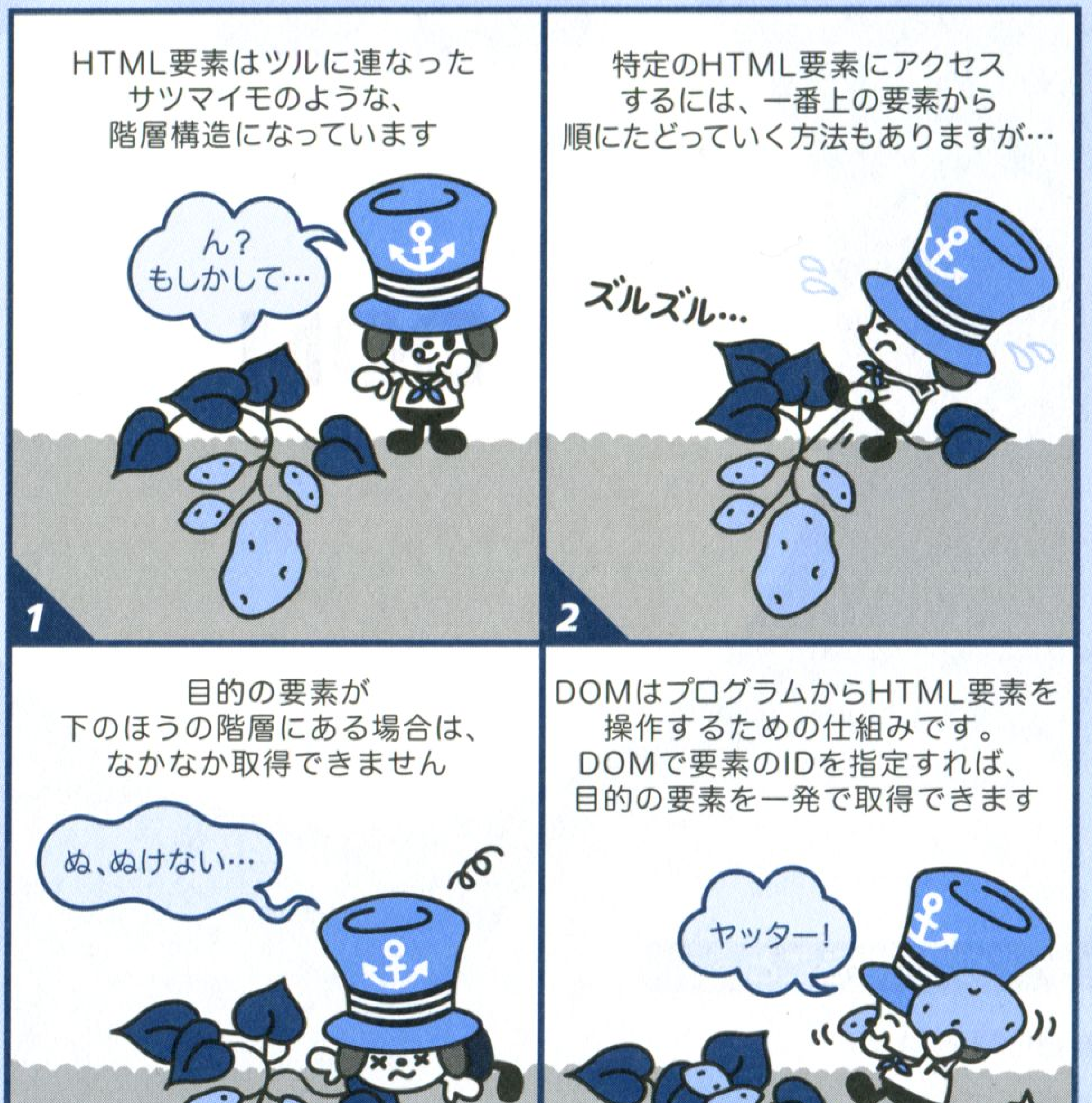
DOM

この章で学ぶこと

- DOMとは
- 要素の取得
- 属性やスタイルの変更
- HTMLの変更
- イベントハンドラの設定



DOMでHTML要素を操作する



本章では、前章で扱ったDocumentオブジェクトによる要素へのアクセスをより深く理解し、HTMLに対するさまざまな操作方法を学びます。

10

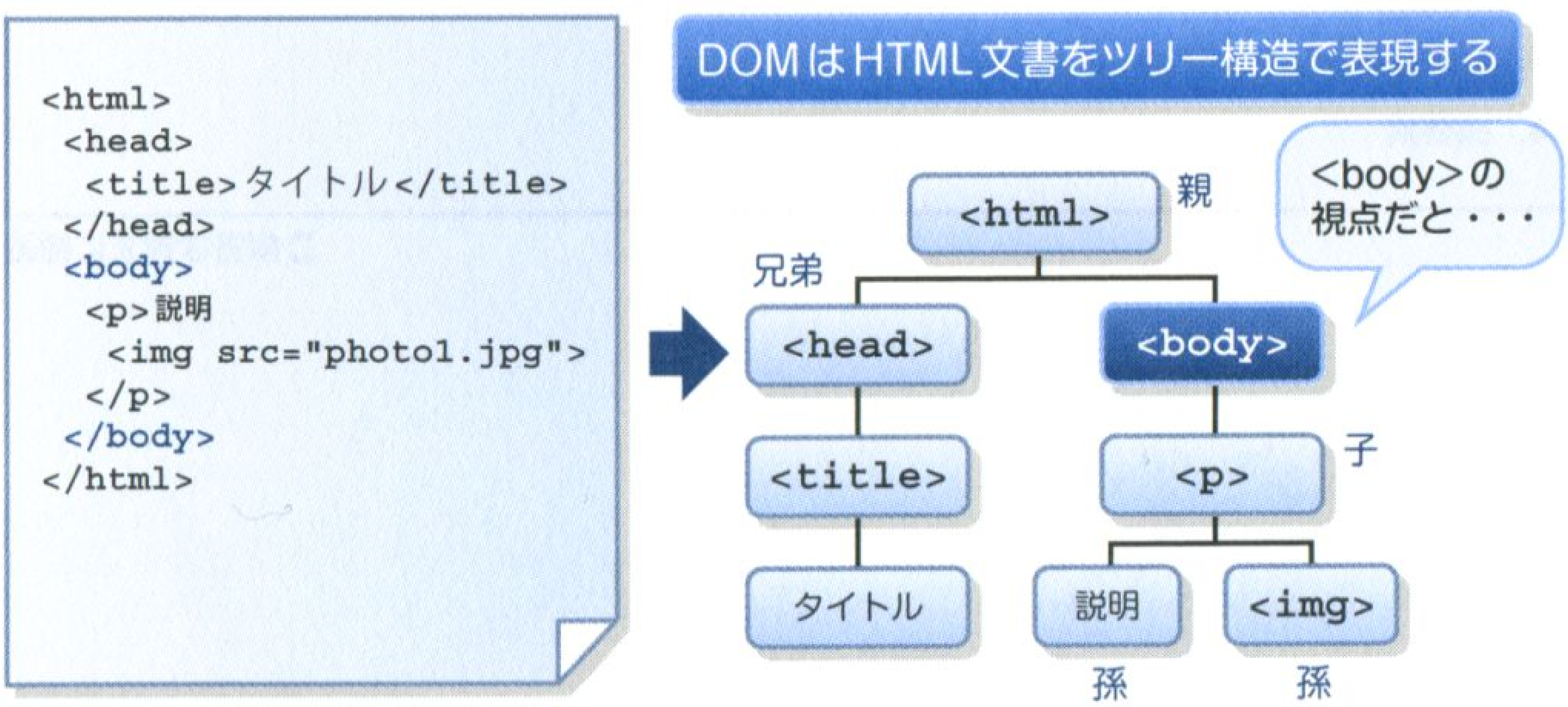
10

1 DOM

DOMとは

DOM（Document Object Model）は、HTMLやXMLなどのマークアップ言語で書かれた文書の各要素にアクセスするための仕組みです。前章で学習したDocumentオブジェクトが持つ要素の構造そのもの、あるいはそれらにアクセスしたり操作したりする機能のことを指します。

DOMは、ドキュメントをツリー構造で扱います。HTML文書の場合、外側のタグを上位の要素として、内側のタグが下に連なっていくイメージになっています。このとき、要素間の関係を親子関係や兄弟関係で表わします。<body>要素視点だと、<html>が親要素、<p>が子要素、<head>が兄弟要素、"説明"と<img>が孫要素になります。



DOMのドキュメント構造

JavaScriptでDOMを扱うには、前節で出てきたDocumentオブジェクトを利用します。Documentオブジェクトには、DOMのツリー構造を利用して要素へのアクセスを容易に行なうメソッドやプロパティが用意されています。次節でそれらを見ていきましょう。

Point!



▶ DOMとは、スクリプトでHTML要素を自由に操作するための仕組みである。

Check Test 1

Q1 以下はDOMに関する説明です。空欄AとBを適切な語で埋めてください

・DOMは、HTMLやXMLなどの［ A ］で記述された文書の要素にアクセスするための仕組みで、要素を［ B ］構造で扱う。

Q2 以下のHTML文書において、<h1>要素から見て<p>要素はどのような関係にあたりますか？

```
<div>
  <h1>タイトル</h1>
  <p>本文</p>
</div>
```

次の中から適切なものを選んでください。

1. 親要素
2. 子要素
3. 兄弟要素
4. 孫要素

➲解答は巻末に掲載

10-2 Documentオブジェクトによる HTML要素の操作

要素へのアクセス（要素の取得）

Documentオブジェクトのメソッドでよく利用されるのは、getElementByIdとgetElementsByNameです。どちらも戻り値としてHTML要素を返します。

文法 IDを指定して要素を取得

```
document.getElementById("id属性値");
```

戻り値 HTML要素

文法 name属性値を指定して要素を取得

```
document.getElementsByName("name属性値");
```

戻り値 HTML要素の集合

id属性にはHTML文書内で一意となる値が設定されるので、getElementByIdメソッドで取得される要素は1つだけです。それに対し、name属性の値は重複が許されているので、複数件の要素が該当する可能性があります。そのため、getElementsByNameメソッドの戻り値は配列と同様の0から始まるインデックスを持ったオブジェクトになります。

注意 getElementsByNameはElementが複数形になっている

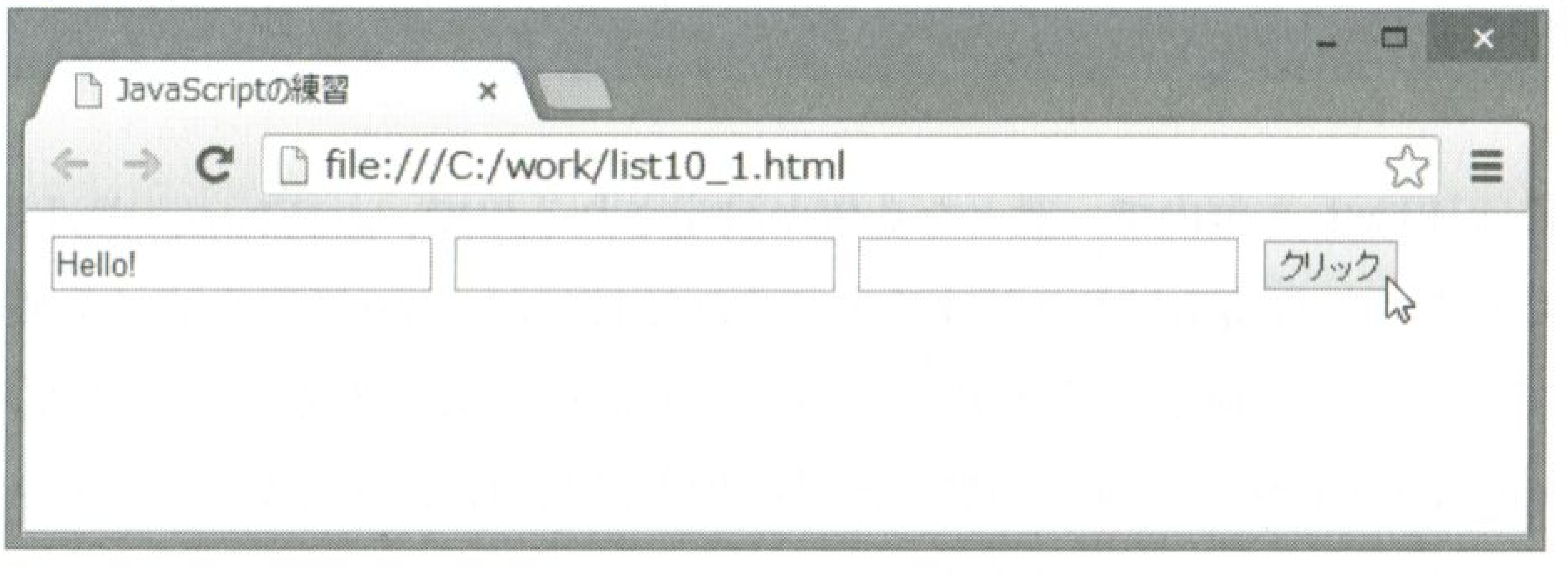
よくある記述ミスとして、getElementByNameと書いてしまう場合があります。「getElement」ではなく「getElements」なので注意してください。getElementsByNameは戻り値として複数の要素を返すので、Element（要素）が複数形になっているのです。

まずはgetElementByIdの利用例から見ていきましょう。

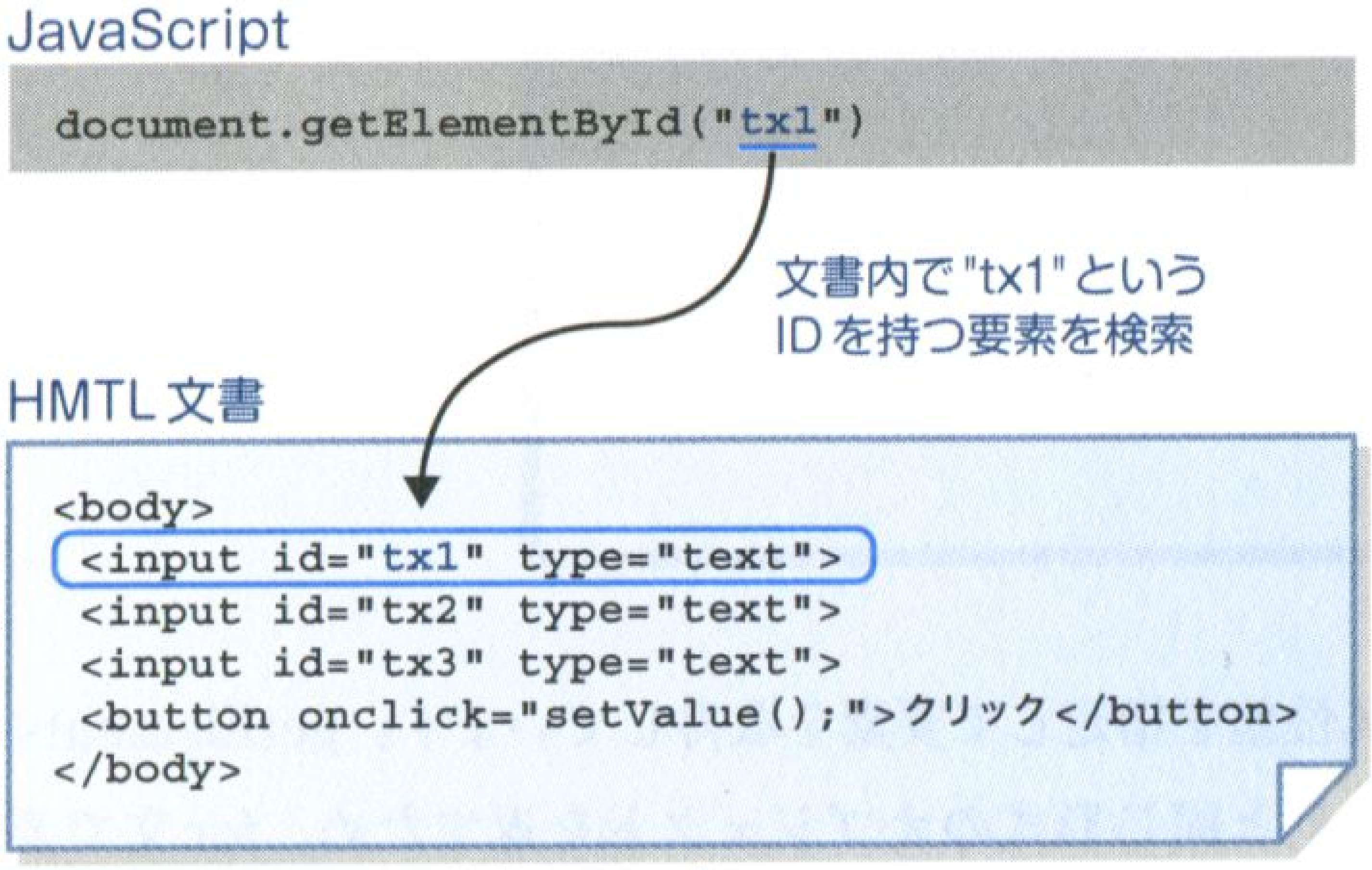
リスト10.1 getElementByIdの利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function setValue() {
     var elm = document.getElementById("tx1");
     elm.value = "Hello!";
   }
  </script>
 </head>
 <body>
  <input id="tx1" type="text">
  <input id="tx2" type="text">
  <input id="tx3" type="text">
  <button onclick="setValue();">クリック</button>
 </body>
</html>
```

■実行結果



リスト10.1は、ボタンがクリックされると"tx1"というID値を持つ要素を探し出し、value属性に値を設定するスクリプトです。



getElementByIdの実行イメージ

この例では、getElementByIdで取得した要素を一度変数elmに代入しましたが、次のように変数を使わず1行で記述することも可能です。

```
document.getElementById("tx1").value = "Hello!";
```

Note 以降のサンプルコードでは、<body>要素内とJavaScript（リスト10.1の網掛け部分）のみを掲載します。

続いて、getElementsByNameの利用例を見てみましょう。

リスト10.2　getElementsByNameの利用

HTML　（<body>要素内部のみ抜粋）

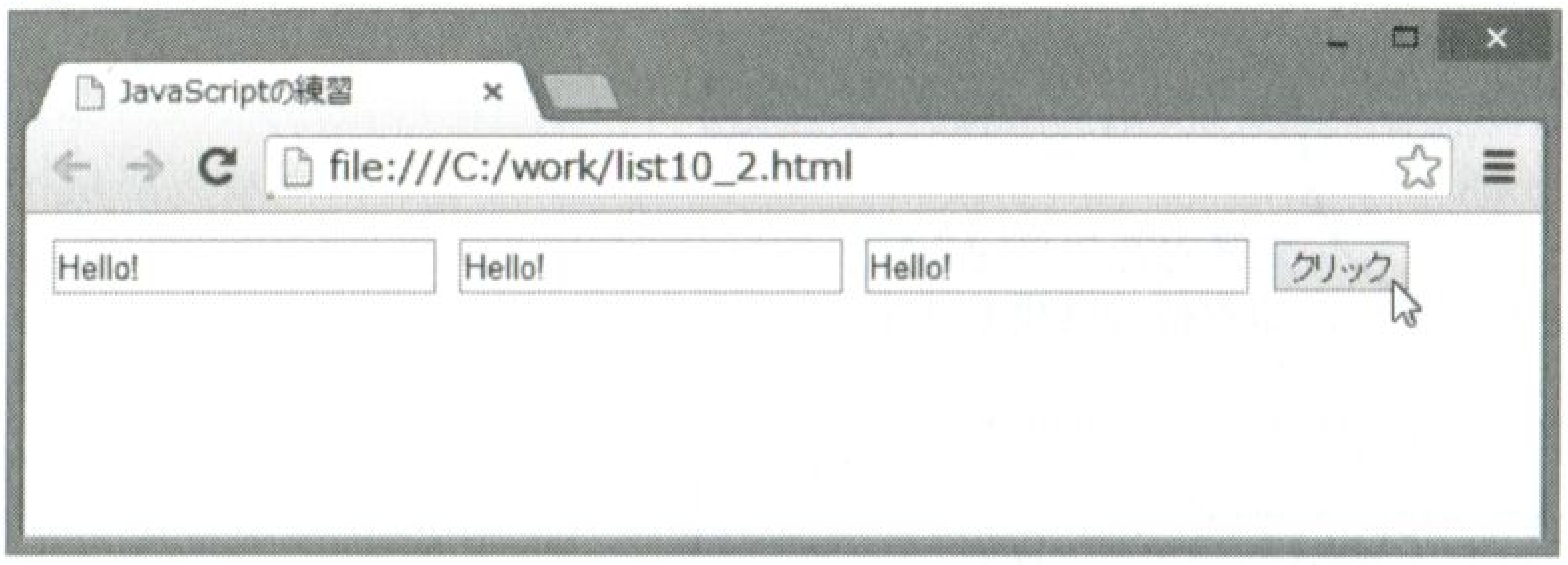
```
<input name="tx1" type="text">
<input name="tx1" type="text">
<input name="tx1" type="text">
<button onclick="setValue();">クリック</button>
```

JavaScript

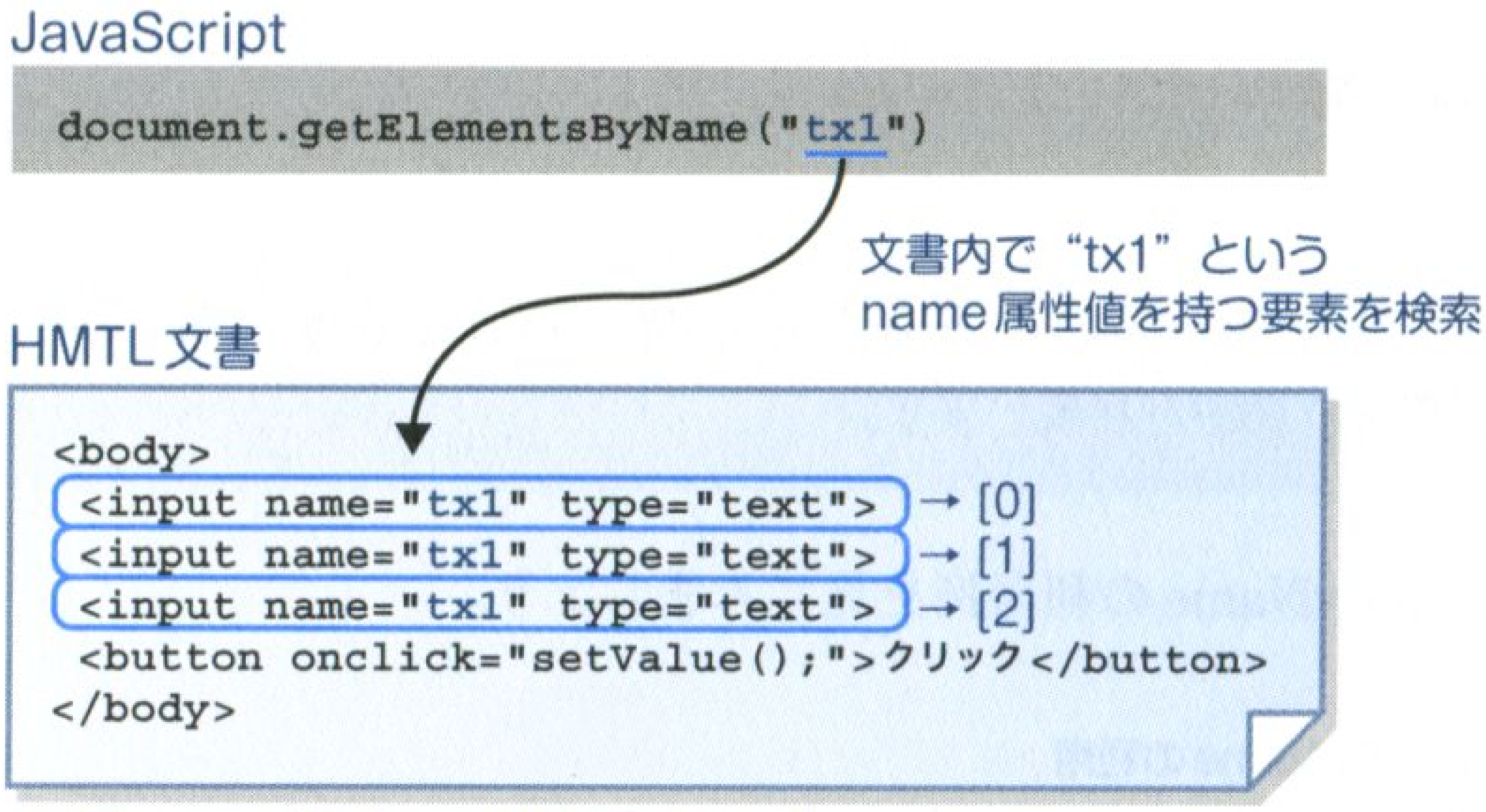
```
function setValue() {
  var elm = document.getElementsByName("tx1");
  for (var i = 0; i < elm.length; i++) {
    elm[i].value = "Hello!";
  }
}
```

10

■実行結果



リスト10.2では、name属性値を指定して要素を取得しています。getElementsByNameは戻り値として配列と同じ形式のオブジェクトを返すため、for文で要素数分だけ繰り返し処理をしています。ちなみに、該当するname属性値を持つ要素が1件だけだった場合でも戻り値は配列形式になるため、インデックスは必ず指定しなければなりません。



getElementsByNameの実行イメージ

◆ その他のHTML属性の操作

value属性に限らず、対象のタグに設定できる属性であれば、スクリプトによる値の変更が可能です。

文法 属性値の変更

```
要素.属性名 = 値;
```

ただし、class属性だけは例外です。classはJavaScriptの予約語候補なので、そのままの名称では使えません。classNameというプロパティを使います。

文法 CSSクラスの変更

```
要素.className = "CSSクラス名";
```

CSSクラスを切り替えることで、ページレイアウトを大幅に変更し、まったく違うWebページのように見せることも可能です。Webページ制作において、レイアウトの動的な変更は不可欠であると言えます。

リスト10.3　表示エリアの切り替え

HTML

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <link rel="stylesheet" type="text/css" href="style.css">
  <script>
   function showIppan() {
     document.getElementById("ippan_area").className = ⏎
"active";
     document.getElementById("hojin_area").className = ⏎
"passive";
   }
   function showHojin() {
     document.getElementById("ippan_area").className = ⏎
"passive";
     document.getElementById("hojin_area").className = ⏎
"active";
   }
  </script>
 </head>
 <body>
  <input type="radio" name="target" value="i" onchange= ⏎
"showIppan();" checked>一般用
  <input type="radio" name="target" value="h" onchange= ⏎
"showHojin();">法人用
  <ul id="ippan_area" class="active">
   <li>氏名<input type="text" name="name"></li>
   <li>住所<input type="text" name="address"></li>
   <li>電話番号<input type="text" name="tel"></li>
  </ul>
  <ul id="hojin_area" class="passive">
   <li>会社名<input type="text" name="company"></li>
   <li>部署名<input type="text" name="busho"></li>
   <li>担当者名<input type="text" name="tanto"></li>
   <li>住所<input type="text" name="address"></li>
```

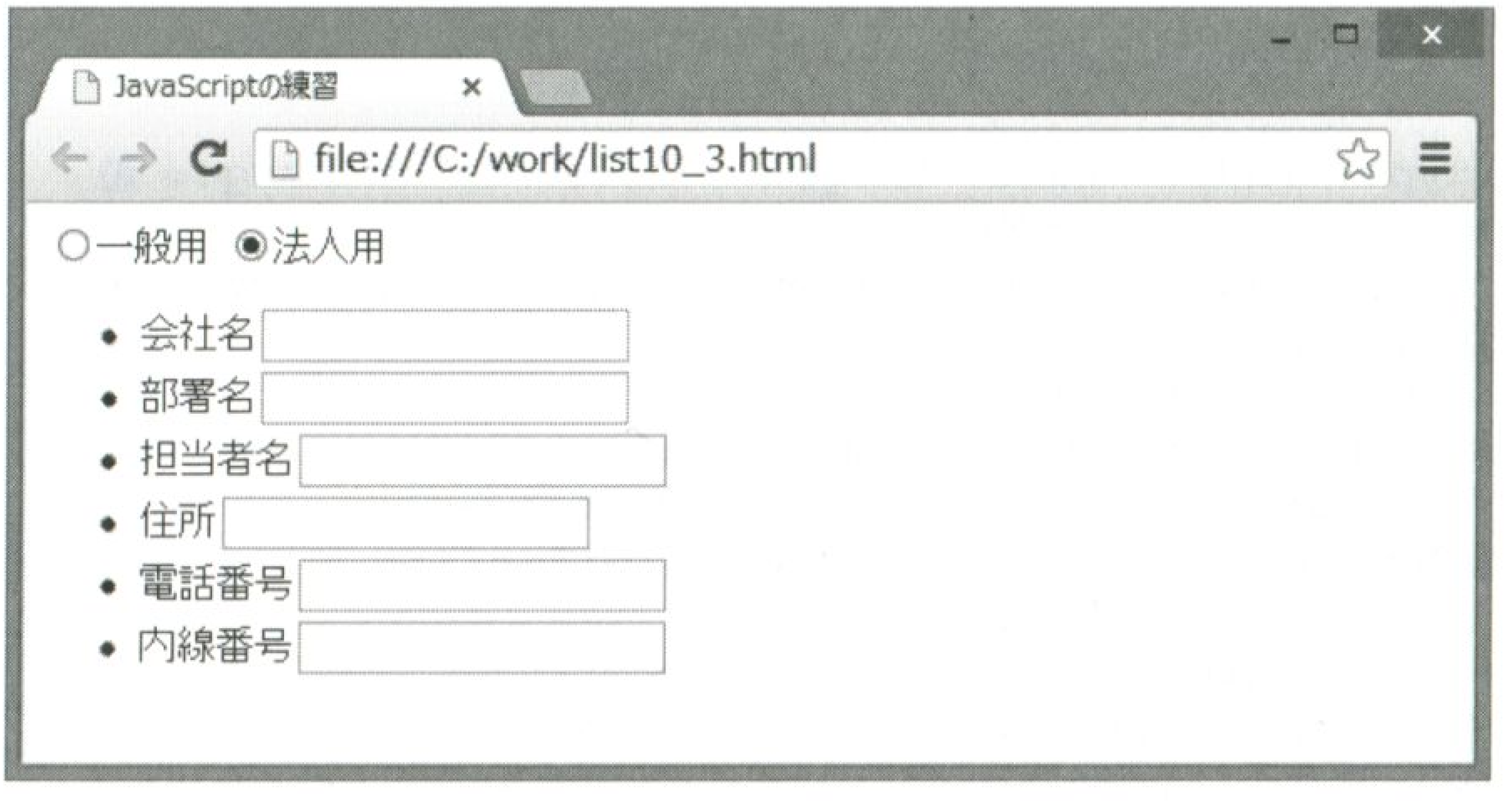
```
    <li>電話番号<input type="text" name="tel"></li>
    <li>内線番号<input type="text" name="naisen"></li>
  </ul>
 </body>
</html>
```

CSS style.css

```
.active {
  display: block;
}
.passive {
  display: none;
}
```

■実行結果

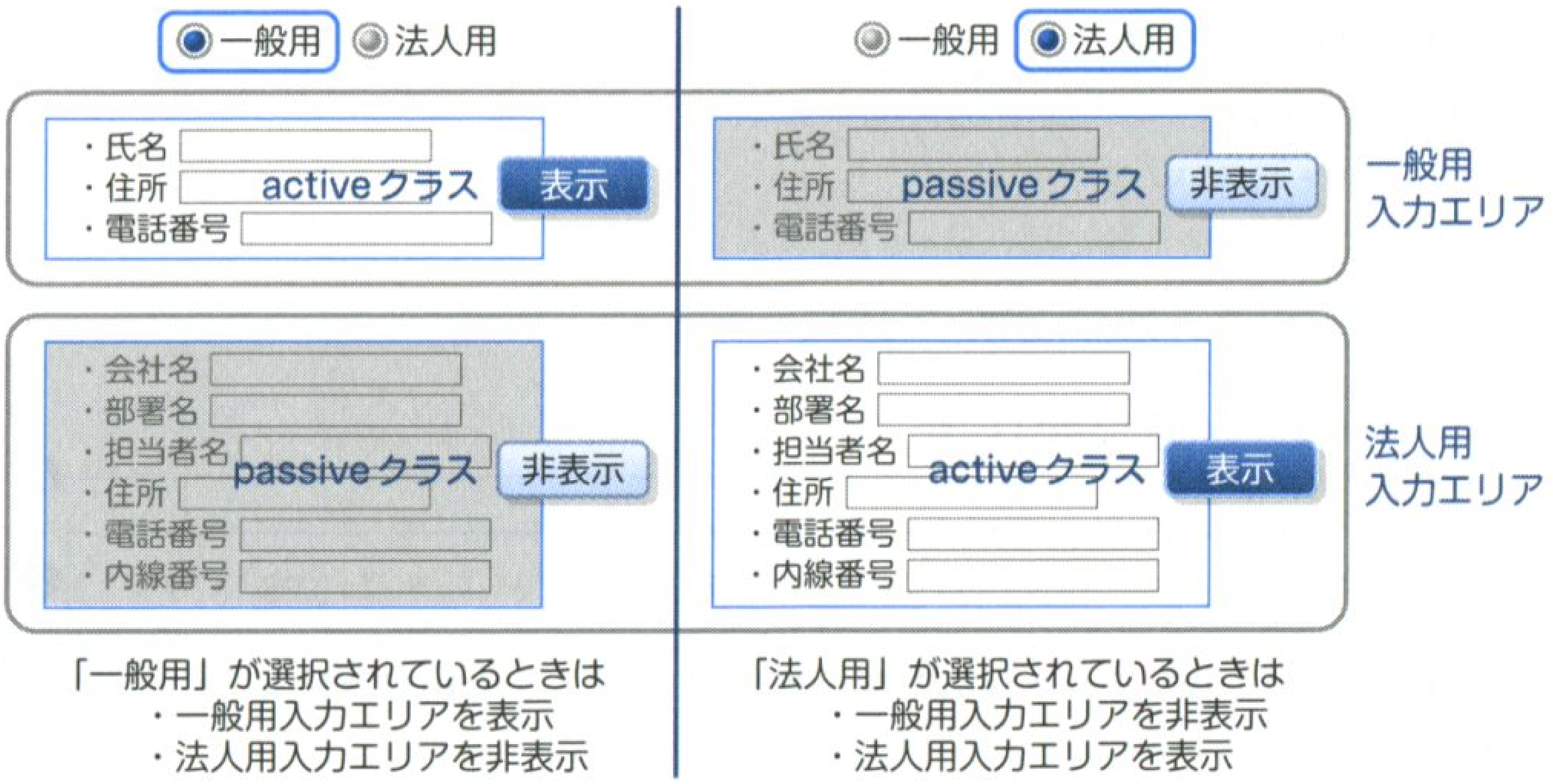




リスト10.3では、選択されたオプションボタンの種類に合わせた入力項目を

表示しています。Webページ上には一般用と法人用、2つの入力項目が存在します。これをJavaScriptでどちらか一方だけ表示するように制御しています。

初期値として一般用入力エリアのclass属性には要素を表示する"active"クラス、法人用入力エリアのclass属性には要素を非表示にする"passive"クラスを設定しています。この2つのCSSクラスを、ラジオボタンの選択状態が変更されたタイミングで交互に切り替えているのです。



クラスによる入力エリアの切り替え

要素の内容の操作

続いて、要素が持つ内容を操作するためのinnerHTMLプロパティを紹介します。

文法 要素の中身を書き換える

```
要素.innerHTML = 文字列;
```

innerHTMLプロパティは、開始タグと終了タグの間に記述されたテキストやタグを表わします。たとえば、

```
document.getElementById("hoge").innerHTML
```

は、以下のHTML文書では囲み部分の範囲を示します。

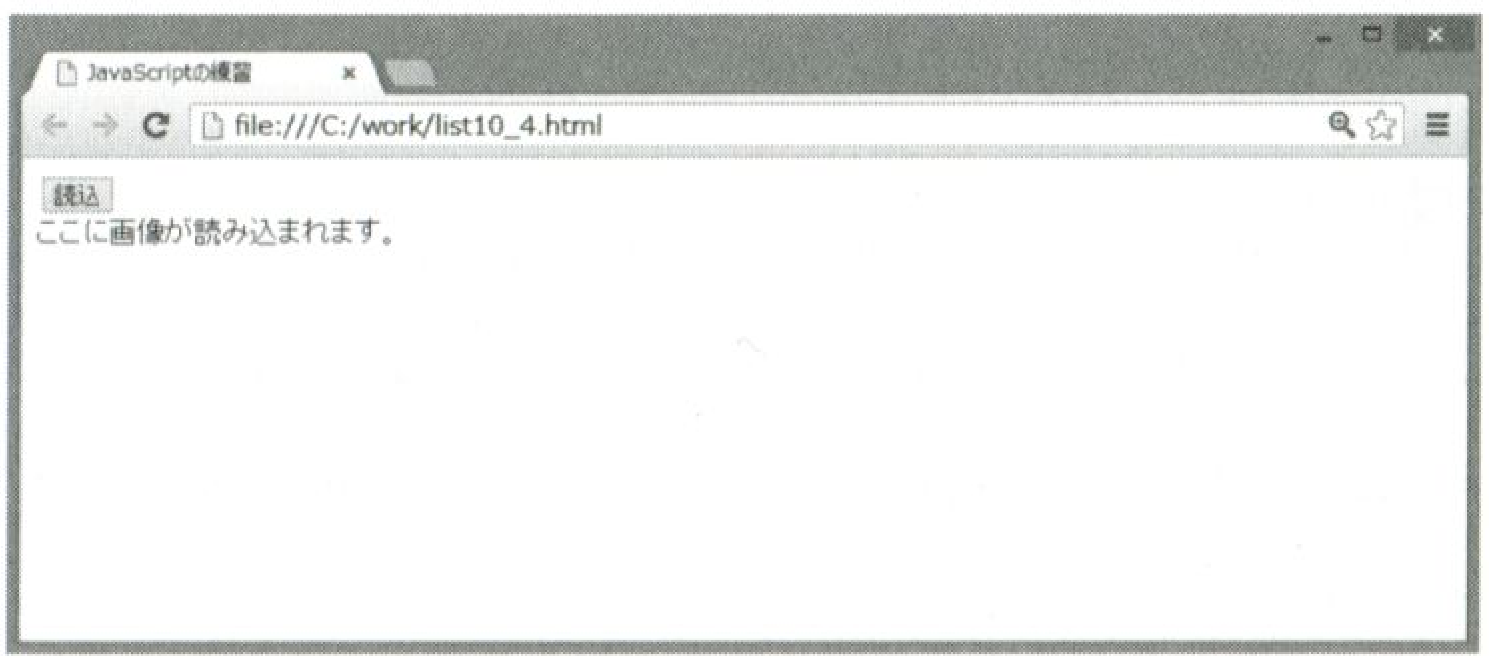
innerHTMLの適用範囲

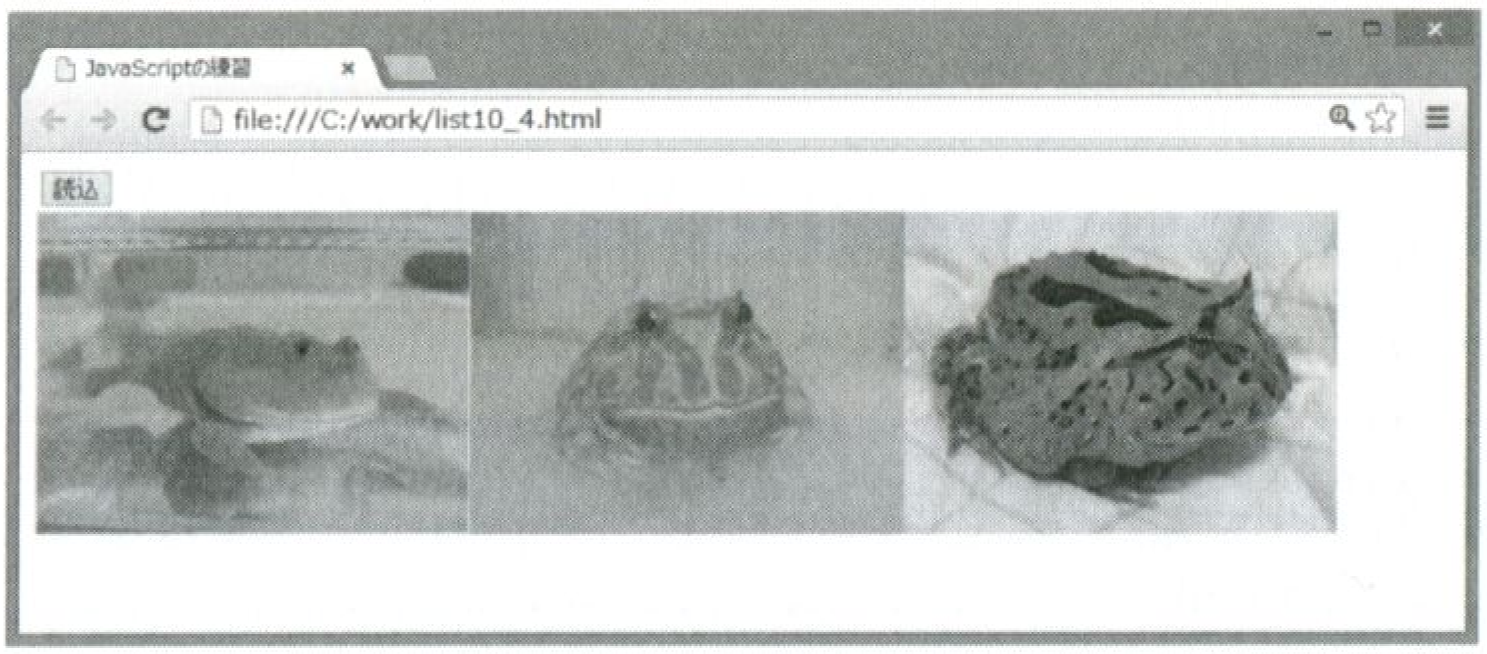
```
<div>
 <p id="hoge">
 文章
 <a href="***.html">リンク</a>      ← innerHTML
 </p>
</div>
```

リスト10.4　画像タグの生成

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function showImage() {
     var htmlstr = "";                                          ❶
     for(var i=1; i<=3; i++) {                                  ❷
       htmlstr += "<img src='images/frog" + i + ".jpg'>";
     }
     document.getElementById("imageArea").innerHTML = htmlstr;  ❸
   }
  </script>
 </head>
 <body>
  <button id="btn" onclick="showImage();">読込</button>
  <div id="imageArea">ここに画像が読み込まれます。</div>
 </body>
</html>
```

■実行結果





リスト10.4では、<div>要素内に3つの<img>要素を挿入しています。showImage関数の処理の流れを確認してみましょう。

まず、リスト10.4の❶の部分で、変数htmlstrを空文字（0文字分の文字列）で初期化しています。これは、この後に続く❷のfor文でhtmlstrに対して文字列の連結処理を行なうためです。変数を初期化しない場合undefinedが既定値になるので、初期値なしのままhtmlstrに文字列を連結すると先頭に"undefined"という文字列が入ってしまいます。

```
var htmlstr = "";
for(var i=1; i<=3; i++) {
  htmlstr += "<img src='images/frog" + i + ".jpg'>";
}
```

❷のfor文の中で、HTML文字列を生成しています。3回分の繰り返しが終了すると、変数htmlstrには以下の文字列が代入されています。

```
<img src='images/frog1.jpg'><img src='images/frog2.jpg'>
<img src='images/frog3.jpg'>
```

最後に、❸の部分で、生成されたHTML文字列を"imageArea"というIDを持つ<div>要素の内容として設定しています。

```
document.getElementById("imageArea").innerHTML = htmlstr;
```

このように、innerHTMLプロパティを利用することで、Webページ上に存在しない部品をスクリプトで作り出すことができるのです。

❶ 変数htmlstrを空の文字列で初期化

htmlstr
```
""
```

❷ for文の中で変数htmlstrにHTML文字列を連結

1回目 htmlstr
```
"<img src='images/frog1.jpg'>"
```

2回目 htmlstr
```
"<img src='images/frog1.jpg'><img src='images/frog2.jpg'>"
```

3回目 htmlstr
```
"<img src='images/frog1.jpg'><img src='images/frog2.jpg'>
<img src='images/frog3.jpg'>"
```

❸ 変数htmlstrの値を<div>要素内にセット

htmlstr
```
"<img src='images/frog1.jpg'><img src='images/frog2.jpg'>
<img src='images/frog3.jpg'>"
```

↓ innetHTMLを置き換える

```
<div id="imageArea">ここに画像が読み込まれます。</div>
```
innerHTML

innerHTMLプロパティの値が設定される流れ

Point!

▶ DOMによって、要素の属性や内容を変更することができる。



■ Check Test 2

Q1 getElementsByNameメソッドの戻り値はどのような形式になっていますか？

Q2 以下の<p>要素の内容を取得し、ダイアログボックスに「こんにちは」と表示するにはどのような命令を記述しますか？

```
<p id="message">こんにちは</p>
```

➲解答は巻末に掲載

10-3 イベントハンドラの設定

JavaScriptによる記述方法

これまで、イベントハンドラはHTMLタグの属性として記述してきました。しかし、HTMLによるコンテンツの定義とJavaScriptによる処理の定義が混在していると、散漫とした見づらい文書になってしまいます。ですので、Webページを制作する際はHTMLとスクリプトを分けて記述することが推奨されています。イベントハンドラの設定もJavaScriptで記述すれば、HTMLとスクリプトを完全に分離させることができます。

HTMLとスクリプトが混在している例

index.html

```
<html>
 <head>
  <script src="common.js"></script>
 </head>
 <body>
  <button id="btn" onclick="sayHello();">クリック</button>
 </body>
</html>
```

HTML中にJavaScriptのコードが混在

common.js

```
function sayHello() {
  console.log("hello");
}
```

HTMLとスクリプトが分離している例

index.html

```
<html>
 <head>
  <script src="common.js"></script>
 </head>
 <body>
  <button id="btn">クリック</button>
 </body>
</html>
```

イベントハンドラの設定もJavaScriptで行なう

common.js

```
function sayHello() {
  console.log("hello");
}
window.onload = function() {
  document.getElementById("btn").onclick = sayHello;
};
```

HTMLとスクリプトの混在と分離

文法 イベントハンドラの変更

```
要素.イベントハンドラ = 定義済みの関数名;
```

または

```
要素.イベントハンドラ = function 関数名() { };
```

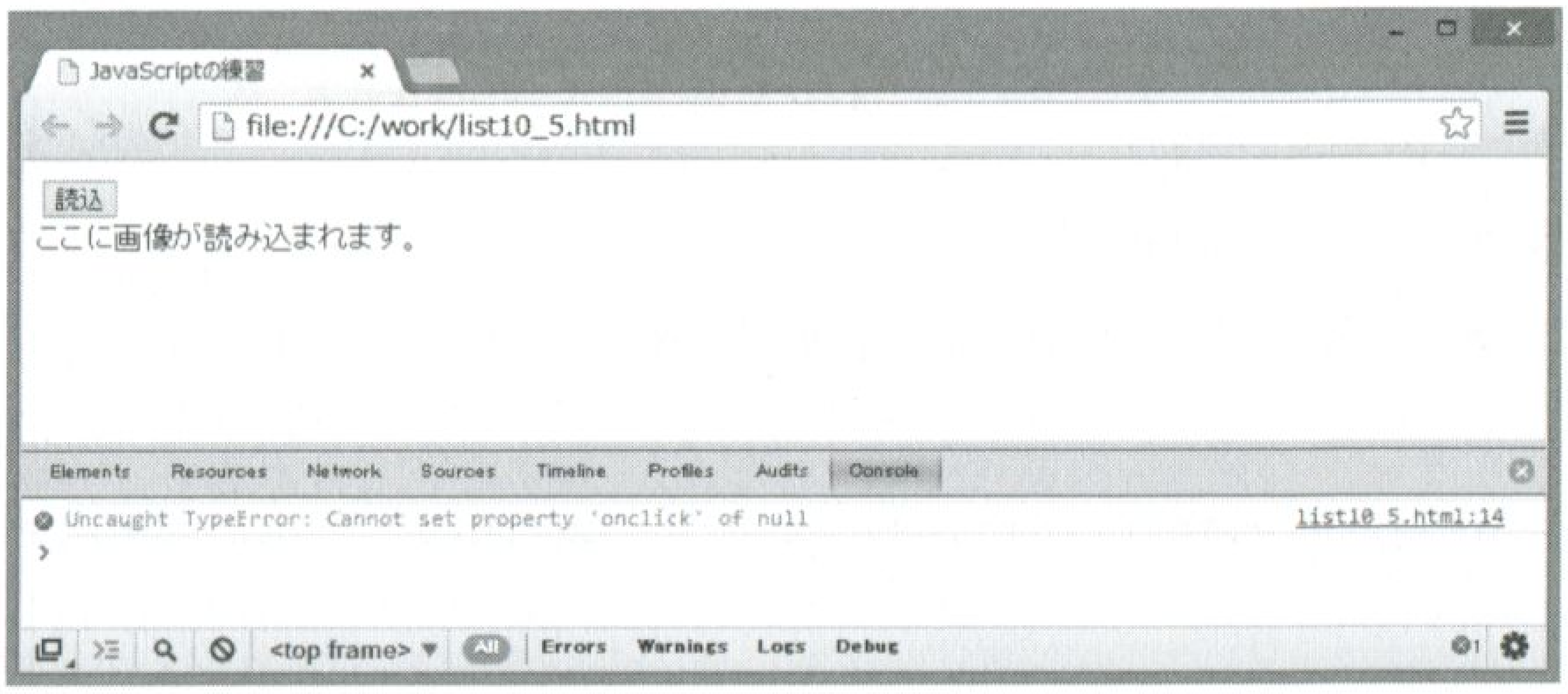
イベントハンドラに関数名、または関数式を代入します。リスト10.5は、リスト10.4を修正してイベントハンドラの設定方法を変更したものです。

リスト10.5 スクリプトによるイベントハンドラの設定

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function showImage() {
     var htmlstr = "";
     for(var i=1; i<=3; i++) {
       htmlstr += "<img src='images/frog" + i + ".jpg'>";
     }
     document.getElementById("imageArea").innerHTML = htmlstr;
   }

   document.getElementById("btn").onclick = showImage;  ❶ 1行追加
  </script>
 </head>
 <body>
  <button id="btn" onclick="showImage();">読込</button>   (onclick="showImage();" 削除)
  <div id="imageArea">ここに画像が読み込まれます。</div>
 </body>
</html>
```

■実行結果



実は、追加した行は不完全なので、実行するとコンソールにエラーメッセージが表示されます。メッセージの文言はブラウザによって異なりますが、「onclickイベントハンドラを設定する対象の要素が存在しない」というような内容のはずです。

HTMLのタグは、基本的に上から順に解釈されていきます。そして<script>タグ内の処理を実行しているときは、以降のタグの読み込みはストップしています。つまり、追加した行の実行段階では、まだ"btn"というID値を持つ要素が読み込まれていない状態なのです。その状態でgetElementByIdメソッド（リスト10.5の❶）を実行すると、要素が見つからないため、戻り値はnull（オブジェクトがないことを表わす値）になります。nullに対してonclickイベントハンドラを設定することはできないので、エラーが発生していたわけです。

```
document.getElementById("btn").onclick = showImage;
```

null（`document.getElementById("btn")` の部分）

<script>タグ内に記述した場合の実行タイミング

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
```

タグは上から順に読み込まれる

```
<script>
  function showImage() {
    var htmlstr = "";
    for(var i=1; i<=3; i++) {
      htmlstr += "<img src='images/frog" + i + ".jpg'>";
    }
    document.getElementById("imageArea").innerHTML = htmlstr;
  }

  document.getElementById("btn").onclick = showImage;
</script>
</head>
<body>
 <button id="btn">読込</button>
 <div id="imageArea">ここに画像が読み込まれます。</div>
</body>
</html>
```

<script>タグ内の処理を実行中はタグの読み込みが中断される

この時点ではIDが"btn"の要素はまだ扱えない

まだ読み込まれていない

◆ onloadイベントハンドラの利用

この問題を解決するには、すべてのタグが読み込まれた後、つまりloadイベントで処理を行なえば良いのです。onloadイベントハンドラは、Windowオブジェクトに対して設定します。

文法 onloadイベントハンドラの設定

```
window.onload = 関数名または関数式;
```

リスト10.5の追加行（❶）を囲むように以下の2行を書き加えると、「ページが表示されたタイミングで、IDが"btn"の要素にonclickイベントハンドラを設定する」という処理が行なわれます。

```
window.onload = function() { ◀------------------------
  document.getElementById("btn").onclick = showImage;  この2行を追加
}; ◀--------------------------------------------------
```

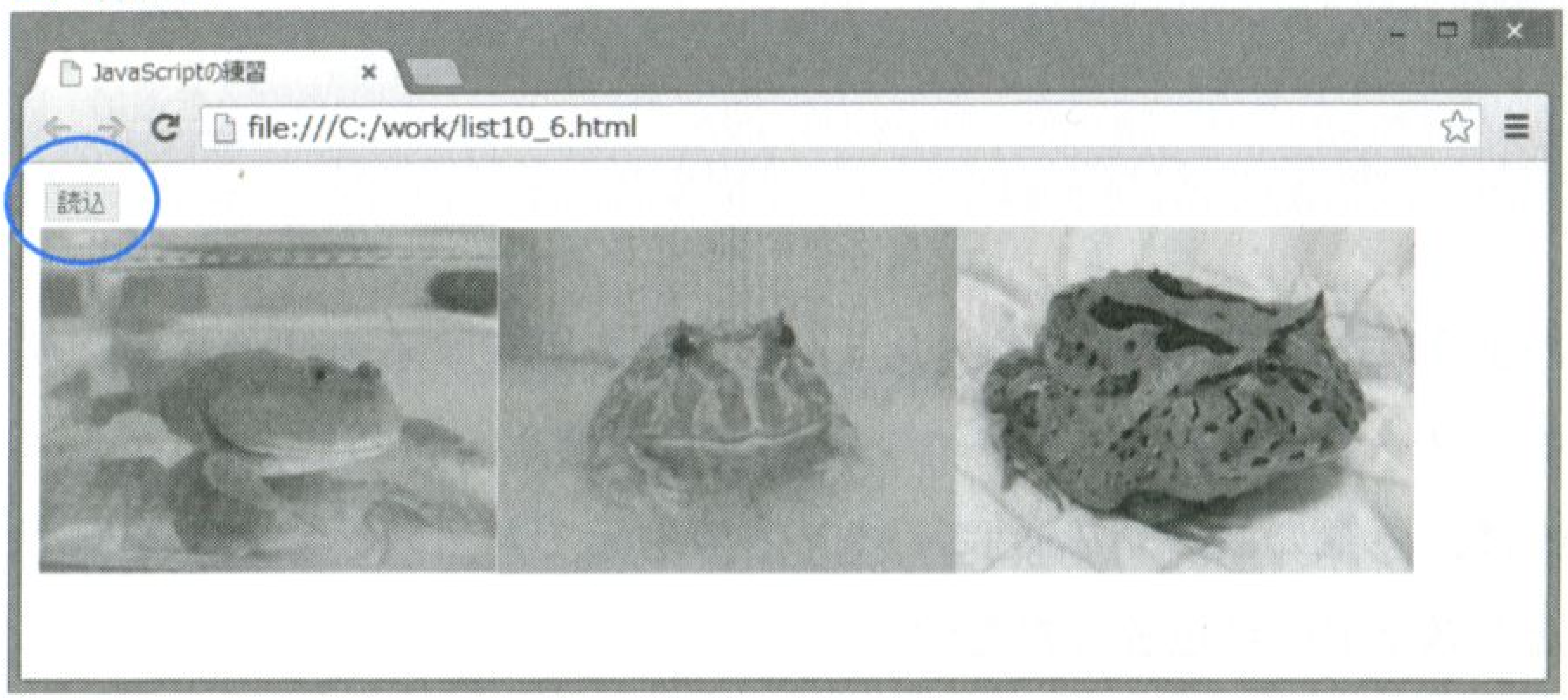
なお、このようにJavaScriptでイベントハンドラを設定する方法の場合、関数内にthisキーワードを記述するとイベント発生元の要素を表わします。

リスト10.6は、画像の読み込み後にイベント発生元である読込ボタンを無効化する処理をリスト10.5に追加したものです。

リスト10.6　イベント処理におけるthisキーワードの利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function showImage() {
     var htmlstr = "";
     for(var i=1; i<=3; i++) {
       htmlstr += "<img src='images/frog" + i + ".jpg'>";
     }
     document.getElementById("imageArea").innerHTML = htmlstr;
     this.disabled = true;  ボタンの無効化処理
   }
   window.onload = function() {
     document.getElementById("btn").onclick = showImage;
   };
  </script>
 </head>
 <body>
  <button id="btn">読込</button>
  <div id="imageArea">ここに画像が読み込まれます。</div>
 </body>
</html>
```

■実行結果



引数のある関数をイベントハンドラに設定する

ここまでのやり方では、引数を持つ関数をイベントハンドラに設定することはできません。仮にリスト10.5のshowImage関数が、以下のような、引数として受け取った番号の写真を表示する関数だったとしましょう。

```
function showImage(number) {
  var htmlstr = "<img src='images/frog" + number + ".jpg'>";
  document.getElementById("imageArea").innerHTML = htmlstr;
}
```

上記の関数をイベントハンドラと関連付けるには、イベントハンドラに対して関数式で匿名関数を代入し、匿名関数の中で引数を付けて呼び出します。引数が必要な関数は、匿名関数で包んであげれば良いのです。

引数がない場合

```
window.onload = function() {
  document.getElementById("btn").onclick = showImage;
}
```

引数がある場合

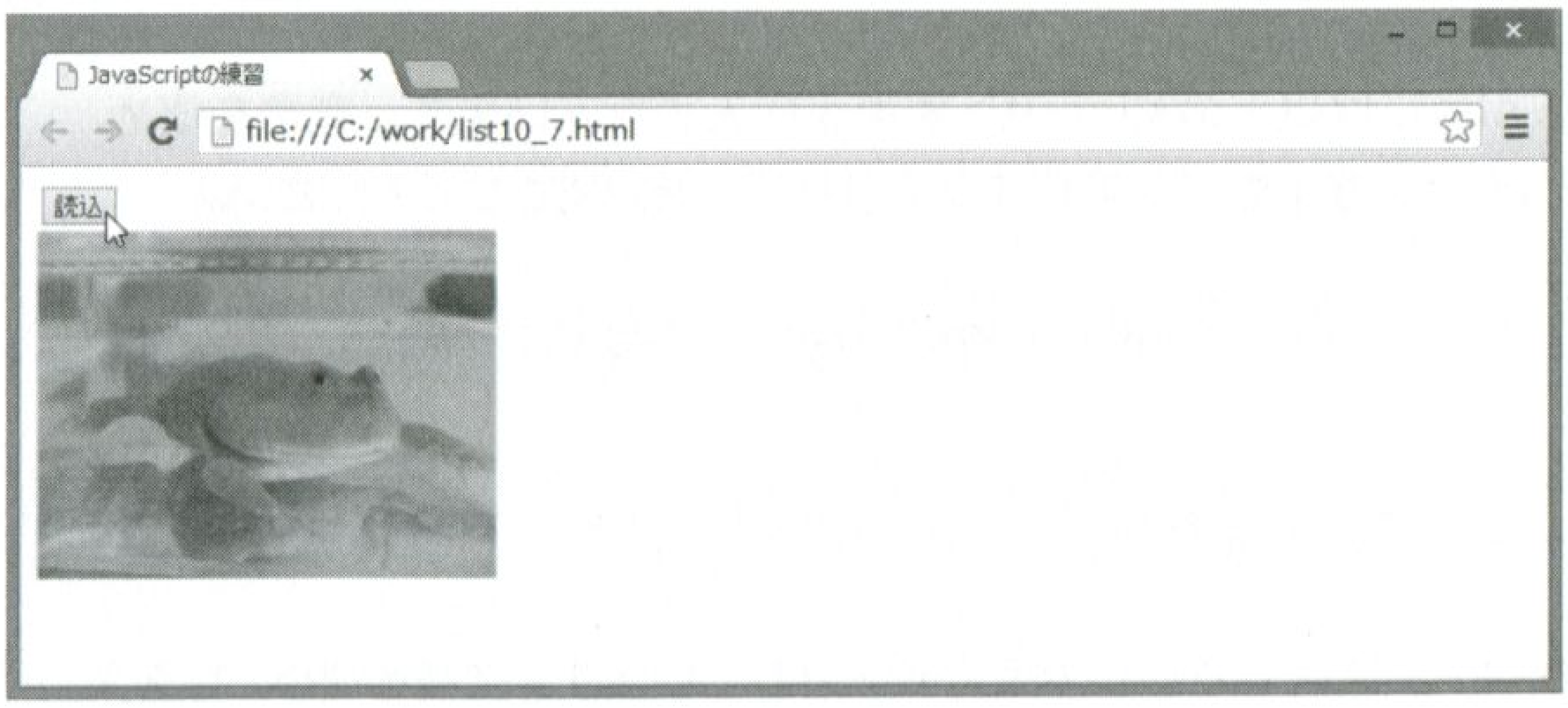
```
window.onload = function() {
  document.getElementById("btn").onclick = function() {
    showImage(1);
  };
};
```

利用例は次のとおりです。

リスト10.7　引数を受け取るイベント処理

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   function showImage(number) {
     var htmlstr = "<img src='images/frog" + number + ".jpg'>";
     document.getElementById("imageArea").innerHTML = htmlstr;
   }
   window.onload = function() {
     document.getElementById("btn").onclick = function() {
       showImage(1);
     };
   };
  </script>
 </head>
 <body>
  <button id="btn">読込</button>
  <div id="imageArea">ここに画像が読み込まれます。</div>
 </body>
</html>
```

■実行結果



高度なDOM操作では、ID以外の方法で要素を取得したり、ページ上に存在しない要素を新たに作成したりすることもできます。詳しくは第12章で学びます。

Point!



▶ HTMLとスクリプトを分離するため、イベントハンドラはスクリプトで設定する。

この章の到達度チェック

Q1 以下の<input>要素のvalue属性に"ABC"を設定します。空欄を埋め、スクリプトを完成させてください。

```
<input type="text" id="hoge" name="fuga">
```

スクリプト

```
document.[        ]("hoge").value = "ABC";
```

Q2 Q1と同様、<input>要素のvalue属性に"ABC"を設定するには、空欄AとBになにを記述すれば良いですか？

```
<input type="text" id="hoge" name="fuga">
```

スクリプト

```
document.[    A    ]("fuga")[B].value = "ABC";
```

Q3 変数elmには以下の<input>要素が代入されています。空欄を埋め、CSSクラスを"right"に変更するスクリプトを完成させてください。

```
<input type="text" id="hoge" name="fuga" class="left">
```

スクリプト

```
elm.[      ] = "right";
```

Q4 変数elmには以下の<p>要素が代入されています。空欄を埋め、姓を入力するテキストボックスの後ろに名を入力するテキストボックスを追加するスクリプトを完成させてください。

```
<p id="p1">
姓<input type="text" name="sei">
</p>
```

スクリプト

```
elm.innerHTML [  ] "名<input type='text' name='mei'>";
```

Q5 ページのloadイベント発生時に、引数の無い関数「func」を呼び出すにはどのようなスクリプトを記述すれば良いですか？　タグの属性として記述する方法は不可とします。

➲解答は巻末に掲載

第11章

jQueryの基礎知識

この章で学ぶこと

- jQueryとは
- クロスブラウザ問題
- jQueryプラグインの利用



● jQueryでらくらくプログラミング ●



昨今、WebサイトでJavaScriptを使う場合は必ずと言って良いほどjQueryとセットで利用されます。まずはjQueryがどんなものなのかを学びましょう。

11 1 jQueryとは

クロスブラウザ問題

JavaScriptが抱える重大な問題として、**クロスブラウザ問題**が存在します。クロスブラウザ問題とは、JavaScriptを実行するブラウザによって、命令の解釈がわずかに異なり、処理結果に相違が生まれてしまうことです。 特に他のブラウザと違う動きをしやすいのが、Internet Explorer（以下、IE）のバージョン8以前です。リスト11.1を見てください。

リスト11.1 子要素のカウント

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script>
   window.onload = func;
   function func() {
     var e = document.getElementById("fruits");
     alert(e.childNodes.length);
   }

  </script>
 </head>
 <body>
  <ul id="fruits">
   <li>りんご</li>
   <li>みかん</li>
   <li>いちご</li>
  </ul>
 </body>
</html>
```

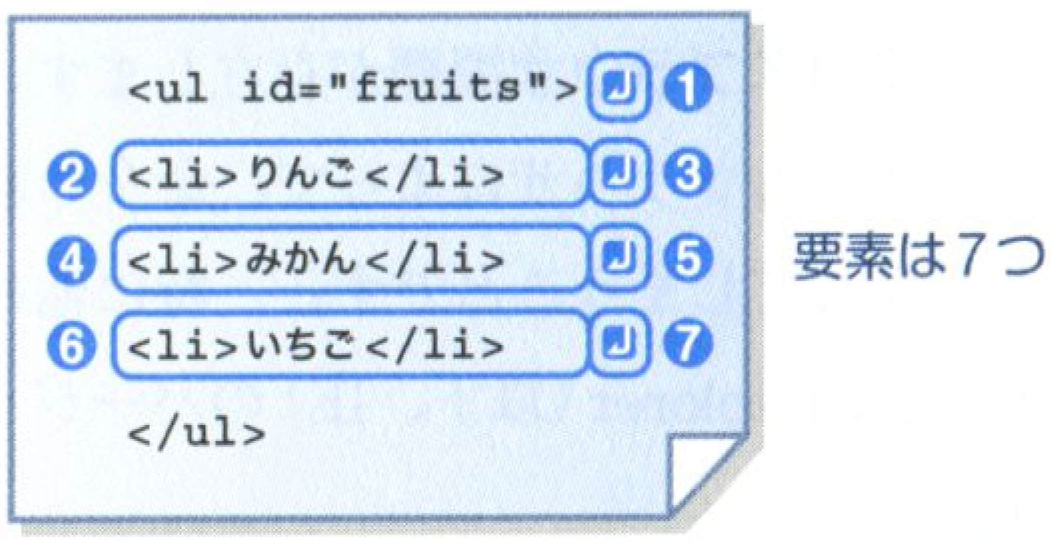
■実行結果 IE8の場合

```
3
```

■実行結果 IE9／Chrome／Firefoxの場合

```
7
```

childNodesプロパティは、要素が持つ子要素を配列形式で表わします。<ul>要素の中には<li>要素が3つあるので、子要素の数は3件だと思うかもしれません。しかし、IE8以外のブラウザでは改行文字も1つのテキスト要素として扱います。<ul>要素の開始タグから終了タグの間までには4つの改行が存在するので、要素数は7件となります。IE8は改行文字をカウントしない仕様になっています。

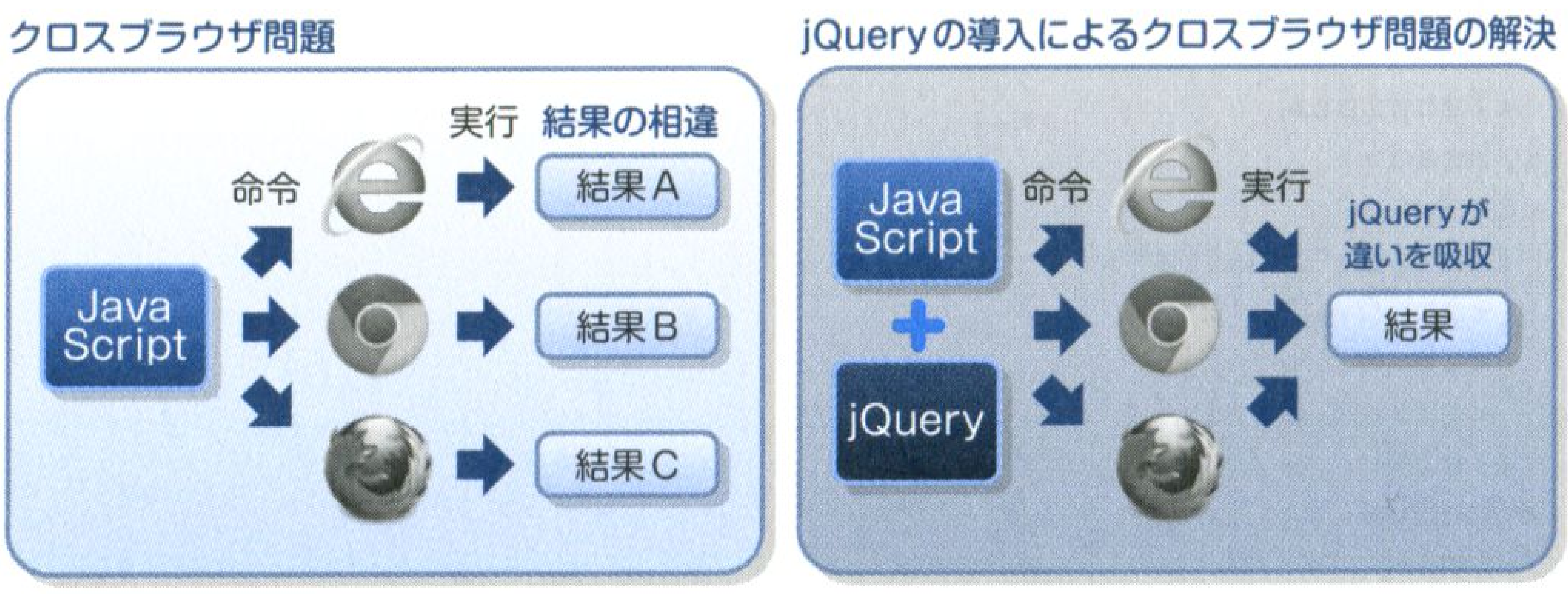


ほとんどのブラウザは改行文字も要素とカウントする

この他にも、ブラウザによって解釈が異なる命令はいくつも存在します。

jQueryのメリット

このクロスブラウザ問題を解決するために生まれたのがjQueryです。jQueryは、DOM操作を提供するJavaScriptのライブラリです。ブラウザ間の違いを吸収してどのブラウザで実行しても同じ処理結果を導いてくれます。



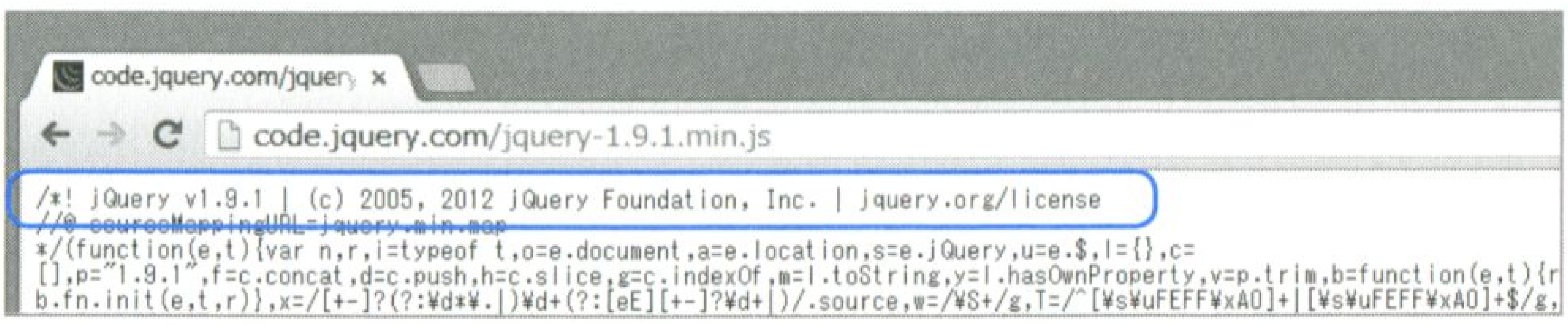
クロスブラウザ問題とjQuery

> **Note** よく利用される機能をまとめて部品化したプログラムをライブラリと呼びます。JavaScriptにおけるライブラリは、ある機能を提供する.jsファイルのことです。

jQueryにはその他にも、以下のようなメリットがあります。

- 文法がシンプルで、JavaScriptを短く記述できる
- CSSの記述方法に似ているので、Webデザイナーにも優しい
- DOMを手軽に利用できる
- Web業界で広く普及しており、JavaScriptを記述する際のスタンダードである
- 商用利用／ソース改変／再頒布可能なオープンソースライセンスである

オープンソースとはソースコードが無償で一般公開されているソフトウェアのことで、ライセンスとは利用者が守るべきソフトウェアの利用条件です。jQueryには、オープンソースライセンスの中でも比較的制限のゆるいMITライセンスが採用されています。MITライセンスは無償で使うことができ、商用利用が可能、かつカスタマイズも自由なライセンスです。利用条件としては著作権の表示が必須となっています。著作権表記は、jQueryも含めた多くのライブラリにおいて、ソースコードの先頭にコメントで記載されています。jQueryに限らず、オープンソースライセンスのライブラリを利用する場合は先頭のコメントを消さないようにしましょう。



jQueryのソースコード内の著作権表記

その他のオープンソースライセンスについては章末のコラムも参照してください。

なお、jQueryは頻繁にアップデートされているので、常に最新版をチェック

しておきましょう。

Point!



- jQueryは、JavaScriptのライブラリである。
- jQueryを使うと、クロスブラウザ問題を解消することができる。

■Check Test 1

Q1 ブラウザによってJavaScriptの実行結果が異なる問題をなんと呼びますか？

➲解答は巻末に掲載

11 2 jQueryの利用準備

入手方法

jQueryを利用するには、jQueryを入手しHTMLファイルに読み込む必要があります。その方法には次の2種類があります。

- 事前にjQuery（.jsファイル）をダウンロードしておき、HTMLファイルに読み込む
- JavaScriptの実行の際にGoogleサーバーからjQueryファイル（.jsファイル）を直接HTMLファイルに読み込む

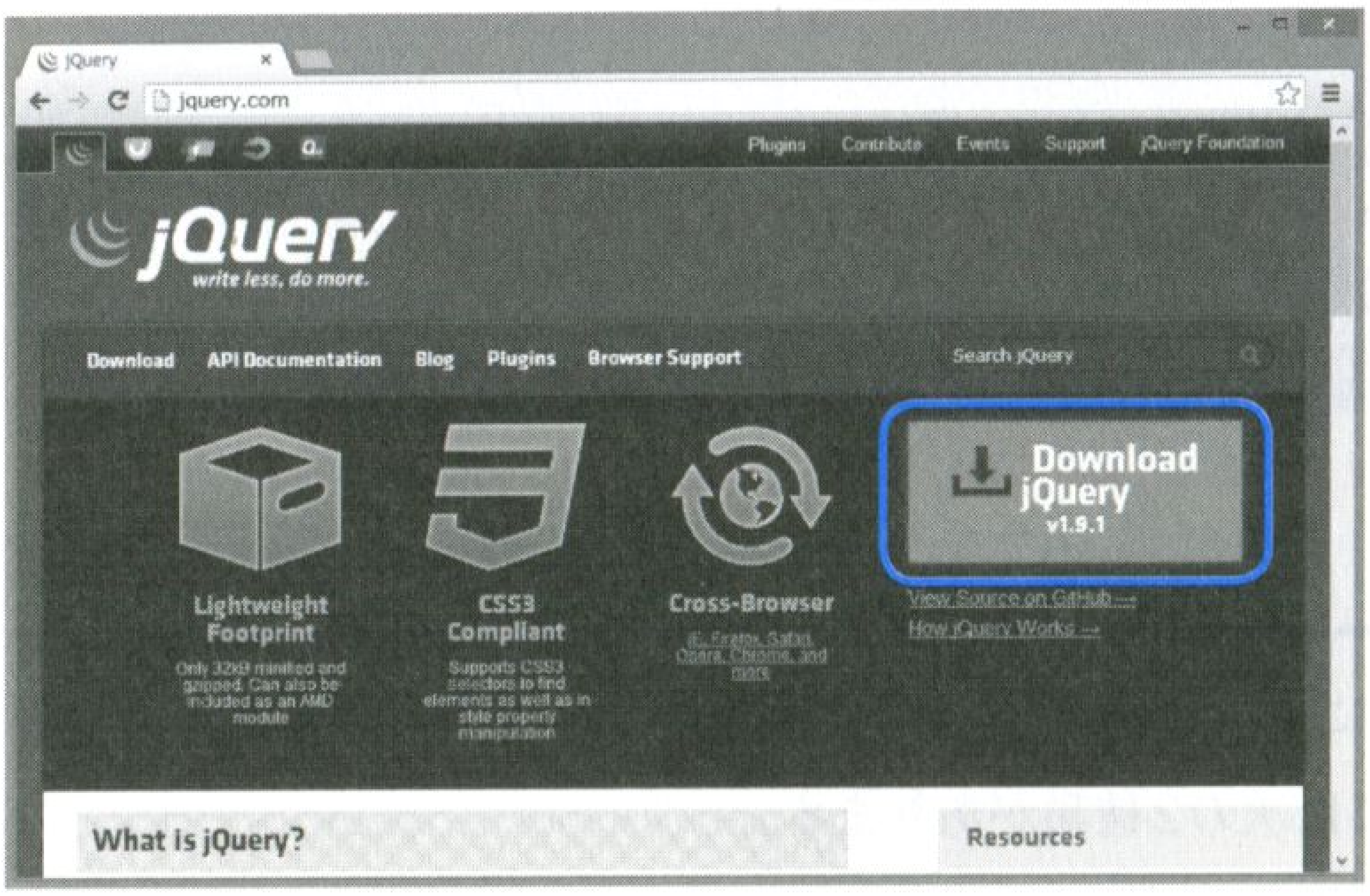
以降で具体的な手順を見ていきましょう。

事前にダウンロードしてHTMLファイルに読み込む

jQueryは、以下からダウンロードできます。自分のパソコン（ローカルPC）上にダウンロードすることで、オフライン環境でもjQueryを実行することができます。

http://jquery.com/

1. ダウンロードボタンをクリックし、さらにDownloading jQueryの「Download the compressed, production jQuery x.x.x」をクリックします（x.x.xにはjQueryのバージョン番号が入ります）。



❷ jQueryのソースコードが表示されるので、ブラウザの機能で「名前を付けて保存」します（Chromeの場合は右クリックして［名前を付けて保存］を選択）。するとjquery-x.x.x.min.jsが保存されます（.jsファイルのx.x.xはjQueryのバージョン番号）。



❸ 保存したjQuery（jquery-x.x.x.min.js）をHTMLファイルに読み込みます。

文法 ダウンロードしたjQueryを読み込み

```
<script src="jquery-x.x.x.min.js"></script>
```

リスト11.1 ダウンロードしたjQueryの読み込み

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <!-- jQueryの読み込み -->
   <script src="jquery-1.9.1.min.js"></script>
 </head>
 <body>
 </body>
</html>
```

Googleのサーバーから直接HTMLファイルに読み込む

ダウンロードをしなくても、Googleのサーバー上に置いてあるjQueryを直接HTMLファイルに読み込んで利用することもできます。この方法を使うと、事前にダウンロードする方法よりもライブラリの読み込み時間が短くなる可能性が高まります。

文法 Googleサーバーからjqueryを読み込む

```
<script src="http://ajax.googleapis.com/ajax/libs/
jquery/x.x.x/jquery.min.js"></script>
```

x.x.xにはjQueryのバージョン番号が入ります。

■記述例

```
<script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
```

以上、2種類のうちのどちらかの方法でjQueryをHTMLファイルに読み込みます。これで、jQueryを使う準備が整いました。

Point!



- jQueryを利用するには、HTMLファイルに<script>要素を使って読み込んでおく必要がある。

■ Check Test 2

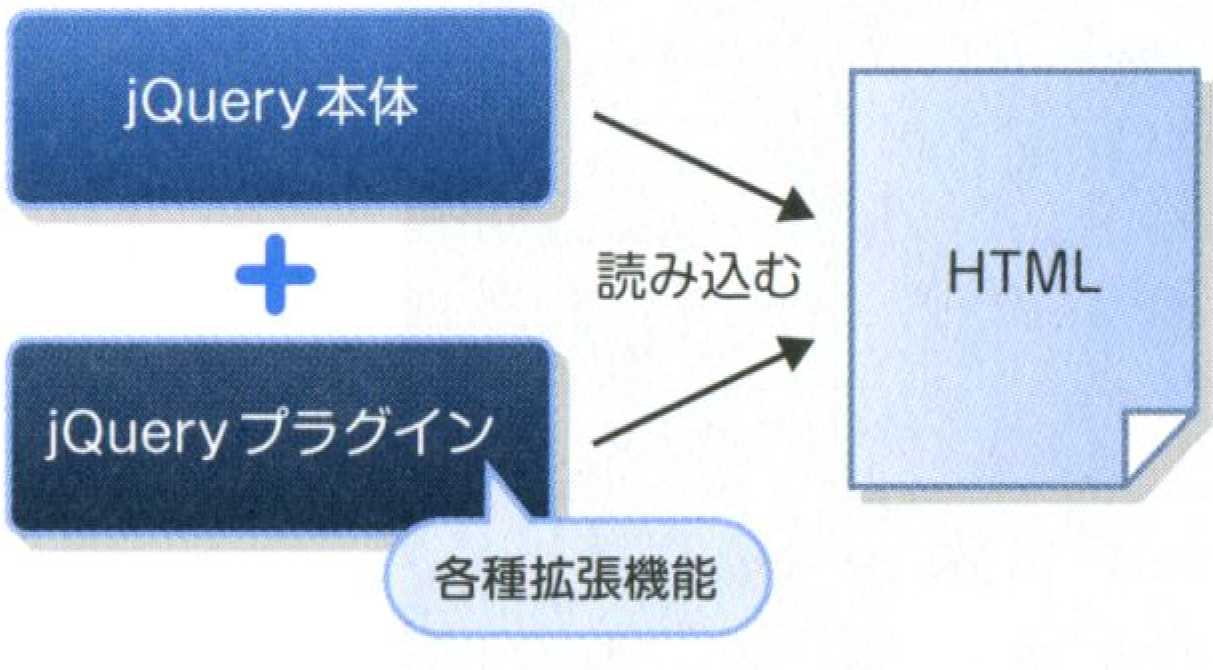
Q1 jQueryを利用するための事前準備として必要なことはなんですか？

➲解答は巻末に掲載

11-3 jQueryプラグイン

プラグインとは

特定のプログラムの拡張機能を**プラグイン**と呼びます。jQueryはさまざまなメソッドを持っていますが、そこにプラグインを組み込むことで、新たなメソッドが追加されます。jQueryに組み込むことができるプラグインをjQueryプラグインと呼びます。jQueryプラグインを利用すれば、自分でプログラミングしなくても、簡単にリッチ（高機能）なWebサイトを構築できます。



jQueryプラグイン

jQueryプラグインは、jQueryの制作者以外でも自由に開発して良いことになっています。現在、有志によってたくさんのjQueryプラグインが開発され、Web上に公開されています。Web検索をすれば使用用途に応じたプラグインが容易に見つかるでしょう。

プラグインの使い方（LightBox系プラグイン）

プラグインは、jQuery本体と一緒にHTMLに読み込んで利用します。ここでは、定番のLightBox系プラグインを例として、プラグインの使い方を学びます。

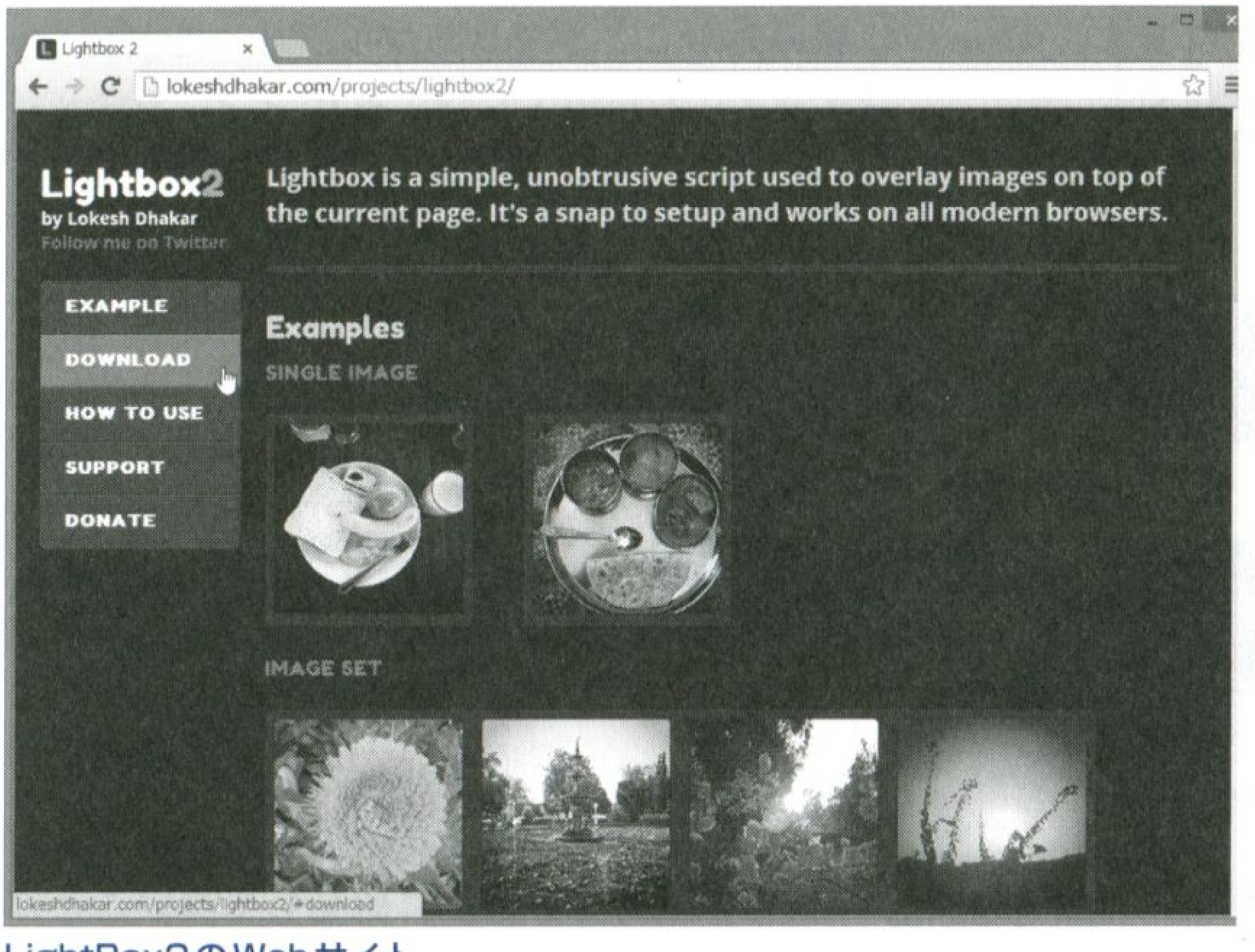
画像をポップアップ表示し、背景をグレーアウトするプラグインを総称して

LightBox系と呼んでいます。その中でも利用方法が簡単なLightBox2の使い方を解説します。

LightBox2は、以下からダウンロードできます。

http://lokeshdhakar.com/projects/lightbox2/

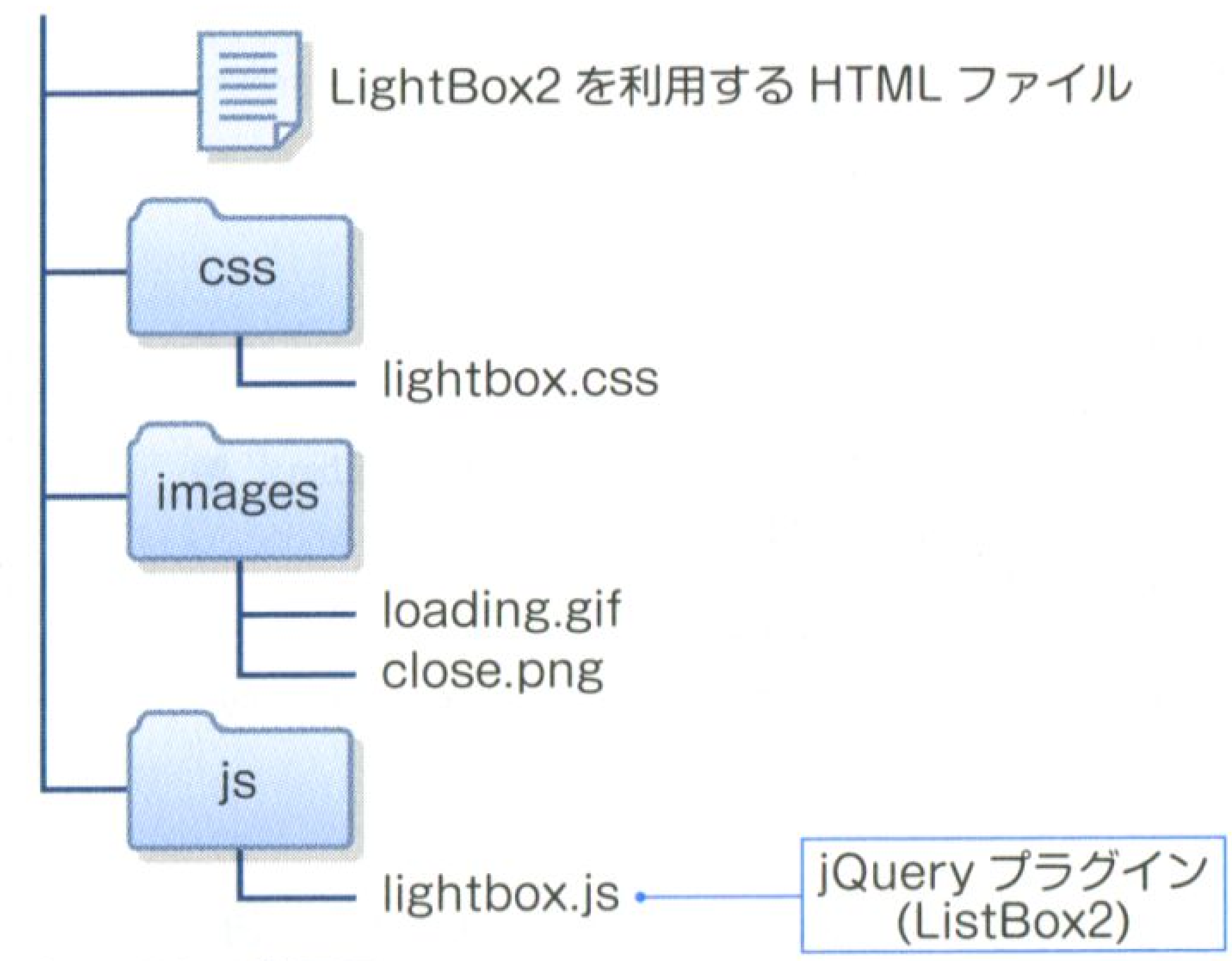
ページ左のDOWNLOADをクリックし、さらにDownloadのボタンをクリックするとzipファイルがダウンロードされます。



LightBox2のWebサイト

ダウンロードしたzipファイルを解凍して現われたファイル群から、最低限必要な以下の4つを、次図のように配置しましょう。

- css/lightbox.css
- images/loading.gif
- images/close.png
- js/lightbox.js



ディレクトリ階層図

まず、jQuery本体と、lightbox.js、lightbox.cssをそれぞれHTMLファイルに読み込みます。jQueryプラグインは、jQuery本体の後に読み込むようにしてください。順番が逆になると、エラーが発生する可能性があります。

```
<script src="http://ajax.googleapis.com/ajax/libs/ ⏎
jquery/1.8.3/jquery.min.js"></script>
<script src="js/lightbox.js"></script>
<link rel="stylesheet" type="text/css" href="css/lightbox.css">
```

続いて、サムネイル画像を<a>タグで囲み、拡大表示する画像のURL、画像タイトルを指定します。rel属性の値は必ず「lightBox」にします。

```
<a href="画像のURL" title="画像タイトル" rel="lightbox">サムネイル画像</a>
```

注意

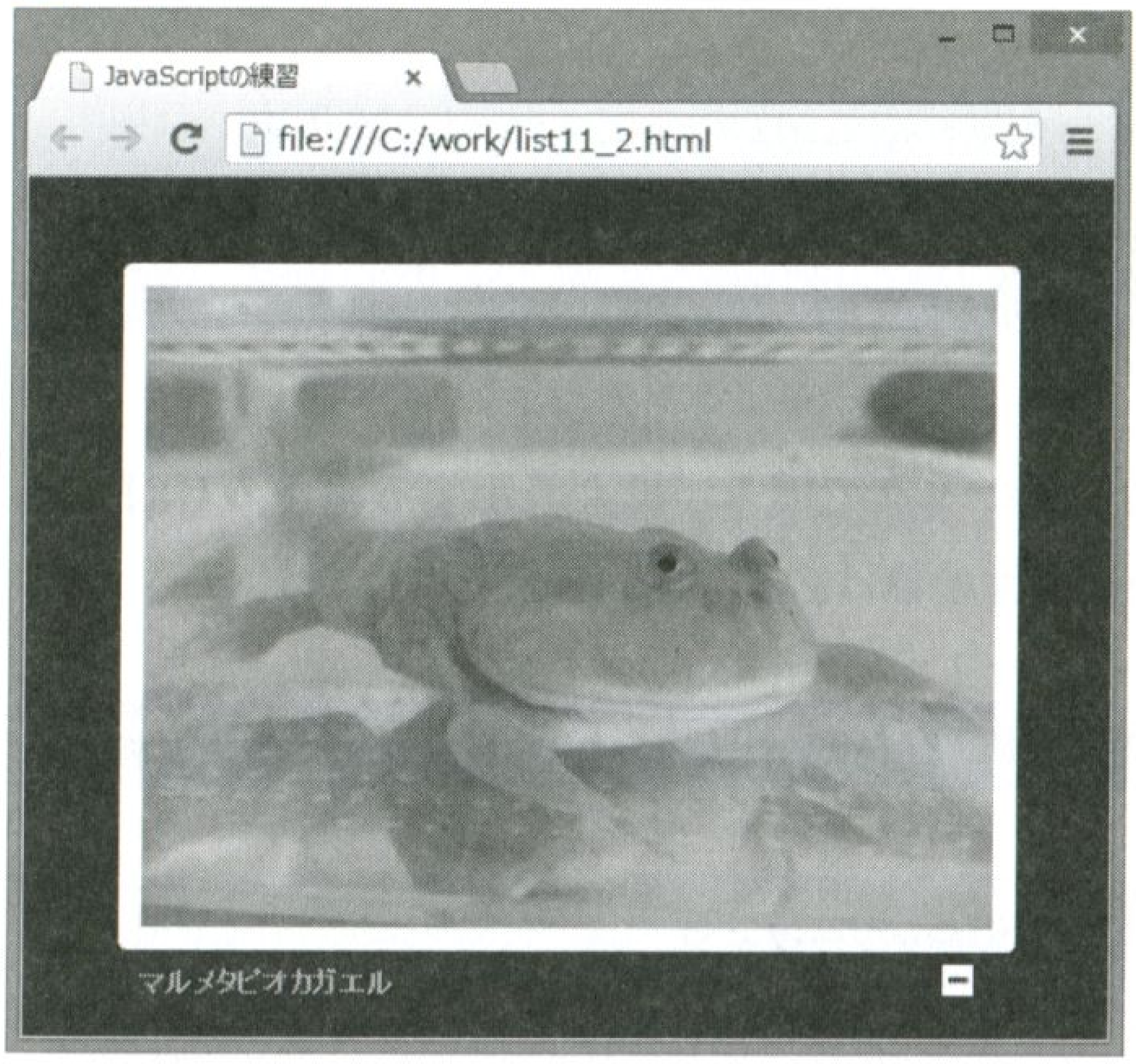
jQuery本体のバージョンとjQueryプラグイン

jQuery1.9.0で過去のバージョンのメソッドが大幅に廃止されたため、jQuery1.9.0のリリース以前に作成されたプラグインを最新バージョンのjQueryと組み合わせると正常に動作しない可能性があります。このサンプル利用しているLightBox2（v2.51）もjQuery1.9.0以降のバージョンに対応していません。そのため、このサンプルではjQuery1.8.3を利用しています。

リスト11.2　LightBox2の利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <!-- jQueryの読み込み -->
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.8.3/jquery.min.js"></script>
  <!-- LightBox2の読み込み -->
  <script src="js/lightbox.js"></script>
  <link rel="stylesheet" type="text/css" href="css/
lightbox.css">
 </head>
 <body>
  <a href="images/frog1.jpg" title="マルメタピオカガエル"
rel="lightbox">
   <img src="images/frog1.jpg" width="20%" height="20%">
  </a>
 </body>
</html>
```

■実行結果（画像クリック後）



Point!

▶ さまざまなjQueryプラグインがWeb上に公開されている。



Column

オープンソースライセンスについて

jQueryやそのプラグインには、さまざまな種類のオープンソースライセンスが設定されています。オープンソースライセンスは一部を除き全般的に次の特徴を持っています。

・無保証である
・商用利用、ソース改変、再頒布が可能

各ライセンスにはそれぞれ遵守すべきルールが細かく定められています。既存のjQueryプラグインに少し手を加えてカスタマイズしたい場合などには、必ずそのプラグインのライセンスを確認しましょう。ここでは代表的なオープンソースライセンスを紹介します。

●BSD License

他のオープンソースライセンスよりも、比較的制限のゆるいライセンスです。著作権表示が義務付けられています。

●MIT License（X11）

BSDと同様に制限がゆるく、著作権表示が義務付けられたライセンスです。jQuery本体はMIT Licenseです。

●GNU General Public License（GPL）

感染性のライセンスです。感染性とは、GPLライセンスのソフトウェアを元に新たなソフトウェアを作成した場合、その新たなソフトウェアもGPLライセンスを引き継ぐ、という性質です。GPLライセンスのライブラリやプラグインをカスタマイズする場合は必ずこの制約を守りましょう。

●Creative Commons License（CC）

４種類の項目を組み合わせて、６段階の制限を付けることができる特殊なライセンスです。LightBox2はこのライセンスです。

項目	概要
Attribution（BY）	利用する際は著作権の表示が必要
Noncommercial（NC）	商用利用は禁止
No Derivative Works（ND）	ソースの改変は禁止
Share Alike（SA）	改変元のCCライセンスを引き継ぐ

これら4項目を組み合わせてライセンスの条件を決定します。組み合わせパターンは次の6種類です。

- BY（著作権表示必須）
- BY-NC（著作権表示必須、商用利用禁止）
- BY-ND（著作権表示必須、ソース改変禁止）
- BY-SA（著作権表示必須、CCライセンスを引き継ぐ）
- BY-NC-ND（著作権表示必須、商用利用禁止、ソース改変禁止）
- BY-NC-SA（著作権表示必須、商用利用禁止、CCライセンスを引き継ぐ）

この章の到達度チェック

Q1 画像をポップアップ表示し、背景をグレーアウトするjQueryプラグインを総称してなんと呼びますか？

Q2 「scripts」ディレクトリの中のjquery-1.9.1.min.jsをHTMLファイルに読み込んで利用する場合、どのようなタグを記述すれば良いですか？

➲解答は巻末に掲載

第12章

jQueryの文法

この章で学ぶこと

- セレクタ
- フィルタ
- 属性の変更
- イベントの設定
- 要素の挿入



jQueryを使うとDOM操作がとっても簡単



jQueryが持つ主な機能は、高度なDOM操作です。複雑なDOM操作も、jQueryを利用すれば簡潔に記述できます。本章で、セレクタやフィルタなど、要素を取得するさまざまな方法を学びましょう。

12 1 jQueryの基本の書き方

readyメソッド

jQueryのコードは、readyというメソッドを使って次の形式で記述します。

文法 jQueryのコード記述❶

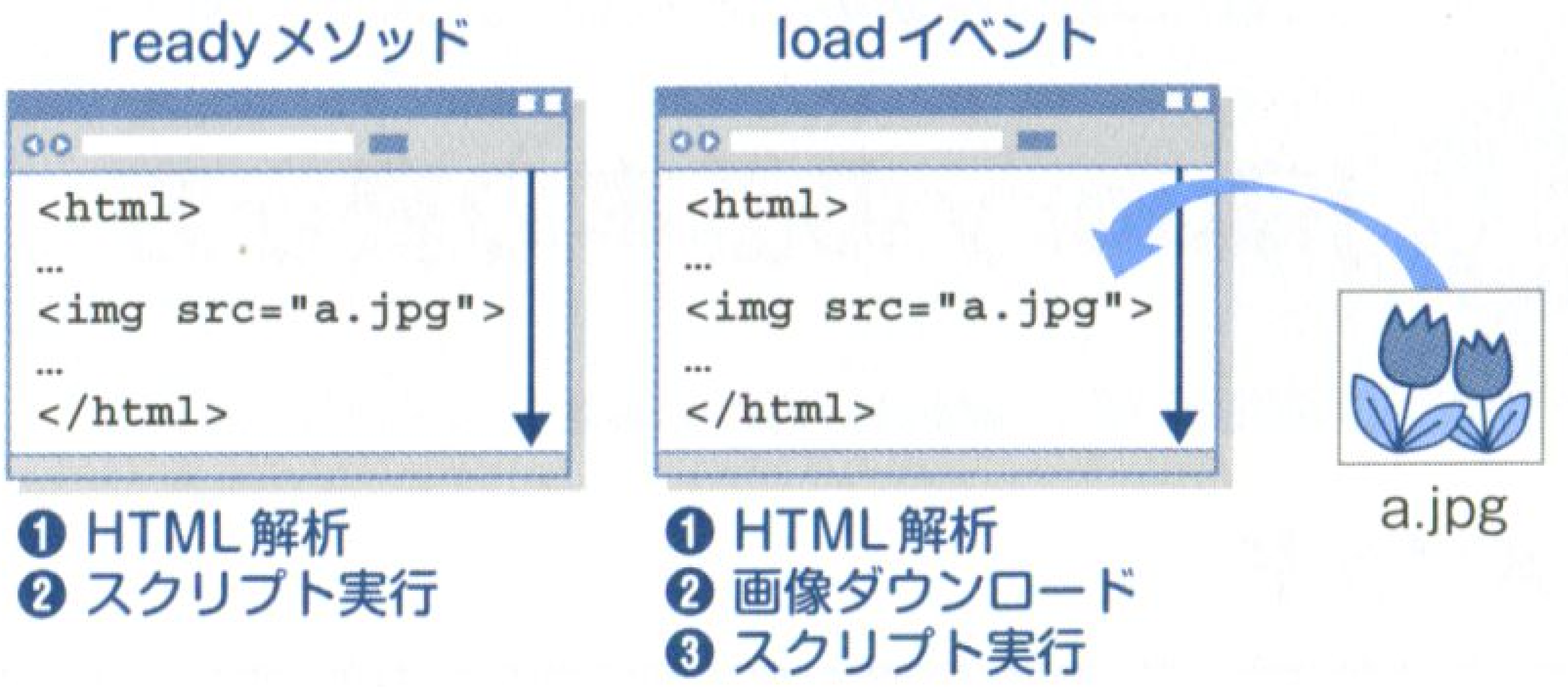
```
$(document).ready(function() {
    // ここにjQueryのコードを記述
});
```

readyメソッドは、HTMLタグが解析され、DOMが利用可能になったタイミングで実行されます。loadイベントの実行タイミングと似ていますが、わずかに異なります。loadイベントはブラウザによって実行されるタイミングに差があり、一部のブラウザでは画像などのデータがすべてダウンロードされた後で実行されます。したがって、大きい画像データがたくさん配置されたWebページでは、loadイベントの発生がかなり遅くなる可能性があります。まとめると表12.1のようになります。

表12.1 ページを開いたときの実行順序

実行順	記述方法	実行タイミング
1	イベントを設定せずにそのまま記述	<script>要素が読み込まれたとき。<body>要素が読み込まれる前
2	readyメソッド	HTMLタグが解析され、DOMが利用可能になったとき
3	loadイベント	HTMLタグが解析され、DOMが利用可能になったとき。一部ブラウザでは、全画像データのダウンロードが終わった後

よって、readyメソッドのタイミングはスクリプトの実行に最適であると言えます。



readyメソッドとloadイベントの実行タイミング

readyメソッドを使った記述方法は長いので、次のように省略して記述することが可能となっています。

文法 jQueryのコード記述❷

```
$(function() {
    // ここにjQueryのコードを記述
});
```

Point!

▶ jQueryのコードは、readyメソッドの中に記述する。

■ Check Test 1

Q1 jQueryのreadyメソッドが実行されるタイミングはいつですか？

Q2 readyメソッドの省略形はどのように記述しますか？

➲解答は巻末に掲載

12 2 セレクタ

要素の取得

jQueryが持つ主な機能は、高度なDOM操作です。jQueryによるDOM操作は、次の手順で行ないます。DOM要素の取得は、セレクタを使って行ないます。

1. セレクタで要素を取得
2. 取得した要素に対してメソッドを実行

これを言い換えると、

1. どの要素に対して
2. どんな処理を行なうか

ということになります。

文法 DOM操作の基本書式

```
$("セレクタ").メソッド();
```

基本的なセレクタは、表12.2の5つです。

表12.2　基本のセレクタ

名前	書式	対象
要素セレクタ	タグ名	指定されたタグ名の要素を取得する
ID セレクタ	#ID 属性値	指定された ID 属性値を持つ要素を取得する
クラスセレクタ	.CSS クラス名	指定された CSS クラス名を持つ要素を取得する
全称セレクタ	*	すべての要素を取得する
グループセレクタ	セレクタ 1, セレクタ 2 …	複数のセレクタのうちいずれかに該当する要素を取得する

jQueryのセレクタは、CSSの記述方法に準拠しています。それぞれのセレクタの詳しい挙動については、第2章を参照してください。ここではIDセレクタを利用したDOM操作を紹介します。

リスト12.1　IDセレクタの利用

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <title>JavaScriptの練習</title>
  <meta charset="utf-8">
  <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
  <script>
   $(function() {
       $("#msg").css("font-size", "30px");
   });
  </script>
 </head>
 <body>
  <p id="msg">こんにちは</p>
 </body>
</html>
```

■実行結果



リスト12.1は、"msg"というIDを持つ要素を取得し、スタイルを変更するスクリプトです。

cssメソッドは、要素にスタイルを適用します。セレクタで取得した要素が複数件ある場合、すべての要素に対してスタイルを設定します。第2引数を省略した場合は、指定したCSSプロパティに現在設定されている値を返します。

文法 スタイルの変更

```
要素.css("CSSプロパティ名", "値");
```

文法 スタイルの取得

```
要素.css("CSSプロパティ名");
```

リスト12.1は、「msg」というIDを持つ要素のフォントサイズを30pxに設定しています。

基本のセレクタのうち、最も速く要素を検索できるのはIDセレクタです。よって、頻繁にアクセスする要素にはIDを設定しておくと良いでしょう。

> **Note** 以降のサンプルコードでは、<body>要素とJavaScript（リスト12.1の網掛け部分）を中心に掲載します。

階層セレクタ

続いて、基本セレクタを組み合わせてより細かく要素を指定する、4種類の階層セレクタを紹介します。「<div>タグの中の<img>タグにのみスタイルを適用したい」など、ある要素を基準としてさらに要素を絞り込む場合に利用します。

表12.3 階層セレクタ一覧

名前	書式	対象
子セレクタ	親要素 > 子要素	ある親要素を持つ子要素を取得する
子孫セレクタ	祖先要素 子孫要素	ある祖先要素を持つ子孫要素を取得する
隣接セレクタ	前要素 + 後要素	前要素の次に現われる後要素を取得する
間接セレクタ	前要素 ~ 兄弟要素	前要素以降に現われる兄弟要素を取得する

階層セレクタを記述するとき、それぞれの要素を示すセレクタはどのような組み合わせでもかまいません。たとえば、親要素はIDセレクタで指定し、子要素はクラスセレクタで指定する、といった形式も可能です。

子セレクタ、子孫セレクタ

子セレクタと子孫セレクタは、ある要素の中に含まれる要素を絞り込み検索するセレクタです。

子セレクタの利用例を見てみましょう。

リスト12.2　子セレクタの利用

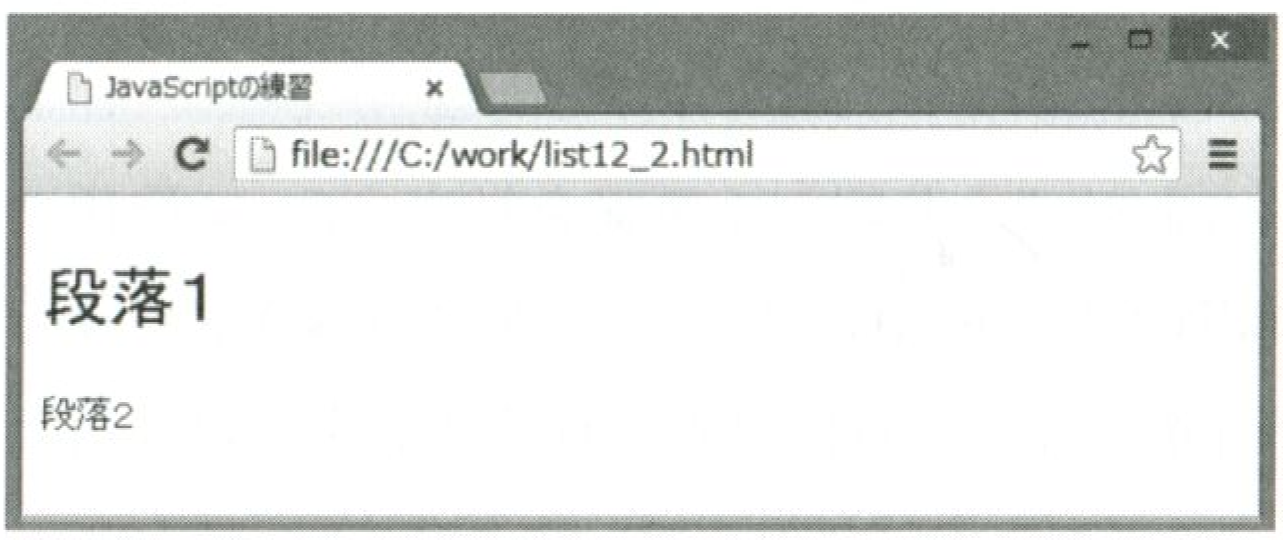
HTML　（<body>要素内のみ抜粋）

```
<div id="parent">
 <p>段落1</p>
 <div>
  <p>段落2</p>
 </div>
</div>
```

JavaScript

```
$(function() {
    $("#parent > p").css("font-size", "30px");
});
```

■実行結果



リスト12.2では、「parent」というIDを持つ要素の子要素うち、<p>要素に対してスタイルを適用しています。次のリストを見てください。子要素1は<p>要素なのでセレクタの対象ですが、子要素2は<div>要素なので対象になりません。また、子要素2の中の<p>要素は孫要素なので、こちらも対象にはなりません。

子要素と孫要素

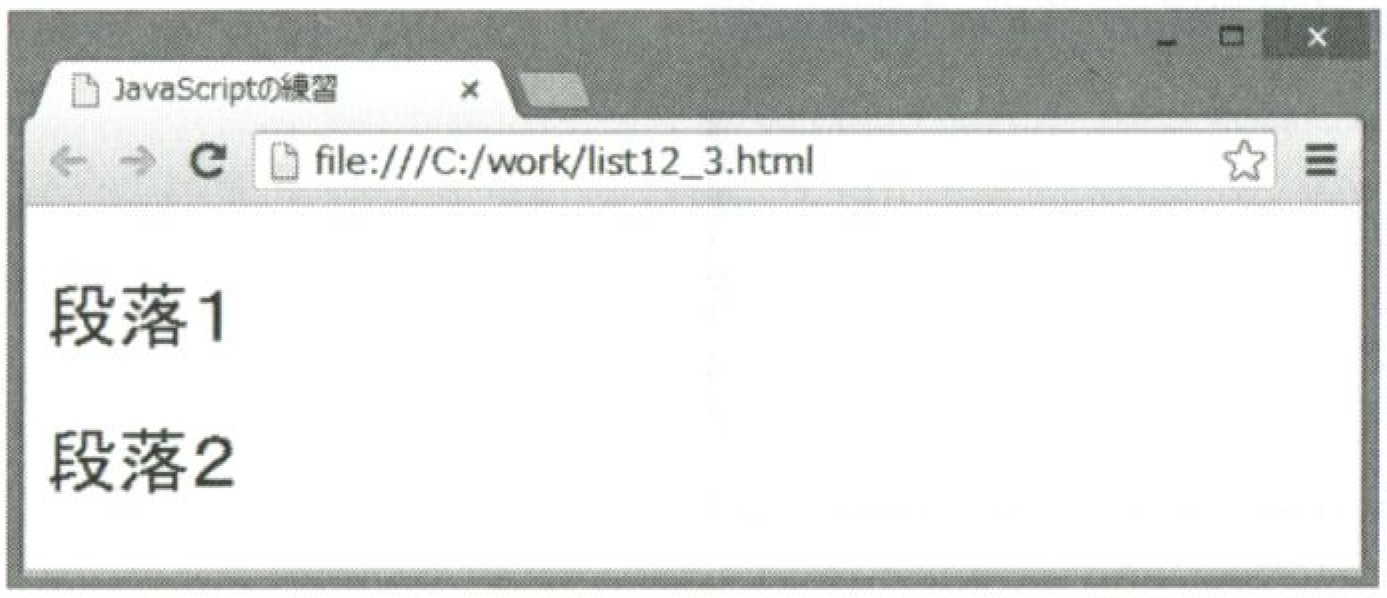
```
<div id="parent">
  <p>段落1</p>          子要素
  <div>
    <p>段落2</p>        孫要素
  </div>
</div>
```

子孫セレクタの場合は、孫以下の要素も対象となります。リスト12.2のセレクタから「>」を削除して実行してみてください。

リスト12.3 子孫セレクタの利用

```
$("#parent p").css("font-size", "30px");
```

■実行結果



「parent」というIDを持つ要素内の<p>要素すべてにスタイルが適用されました。

```
<div id="parent">
  <p>段落1</p>          子要素
  <div>
    <p>段落2</p>        孫要素
  </div>
</div>
```

◆ 隣接セレクタ、間接セレクタ

隣接セレクタ、間接セレクタはともに同じ階層に存在する兄弟要素を得るセレクタです。隣接セレクタは次の兄弟要素を1件、間接セレクタは以降に現われる兄弟要素をすべて取得します。

リスト12.4　隣接セレクタの利用

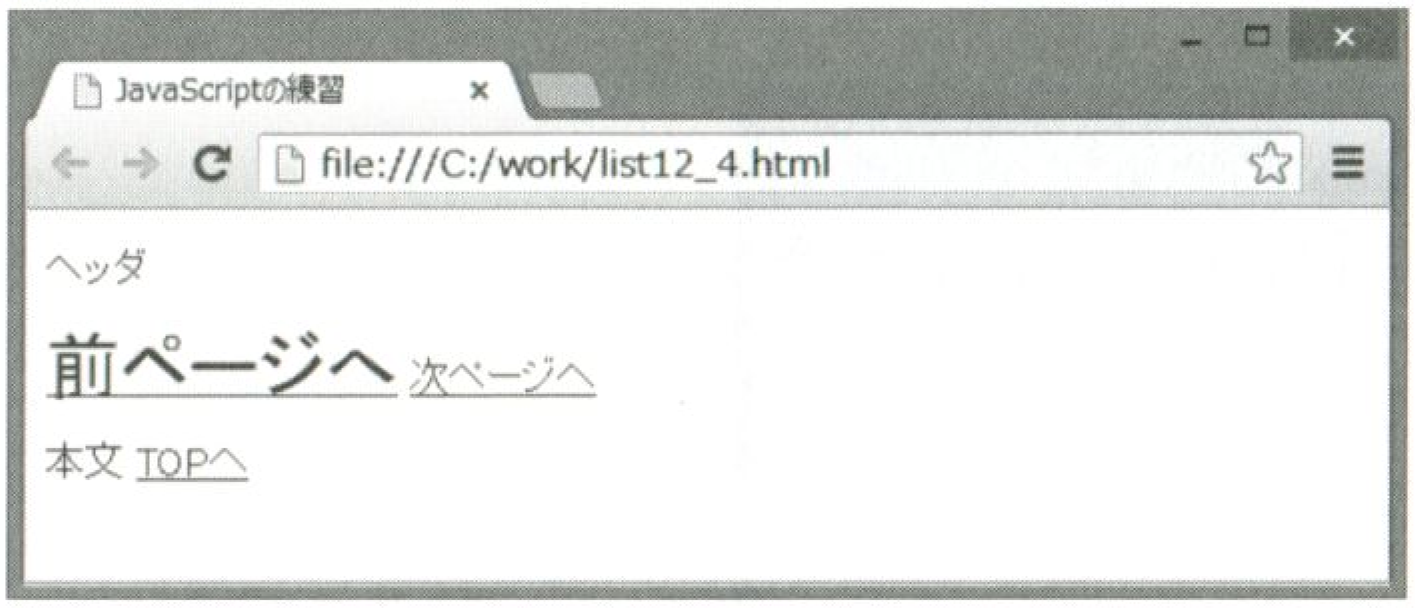
HTML　（<body>要素内のみ抜粋）

```
<p>ヘッダ</p>
<a href="prev.html">前ページへ</a>
<a href="next.html">次ページへ</a>
<p>
  本文
  <a href="index.html">TOPへ</a>
</p>
```

JavaScript

```
$(function() {
    $("p + a").css("font-size", "30px");
});
```

■実行結果



リスト12.4は、<p>要素の次に現われる<a>要素に対してスタイルを適用しています。隣接セレクタが取得する要素は、前要素の直後に現われる要素だけです。「次ページへ」リンクの直前は<a>要素なので対象外です。また、「TOPへ」リンクは<p>要素の兄弟要素ではなく子要素なので、これも対象外となります。

```
<p>ヘッダ</p>
<a href="prev.html">前ページへ</a>      ―― <p>の直後に出現
<a href="next.html">次ページへ</a>
<p>
  本文
  <a href="index.html">TOPへ</a>      ―― <p>の中に出現
</p>
```

リスト 12.5　間接セレクタの利用

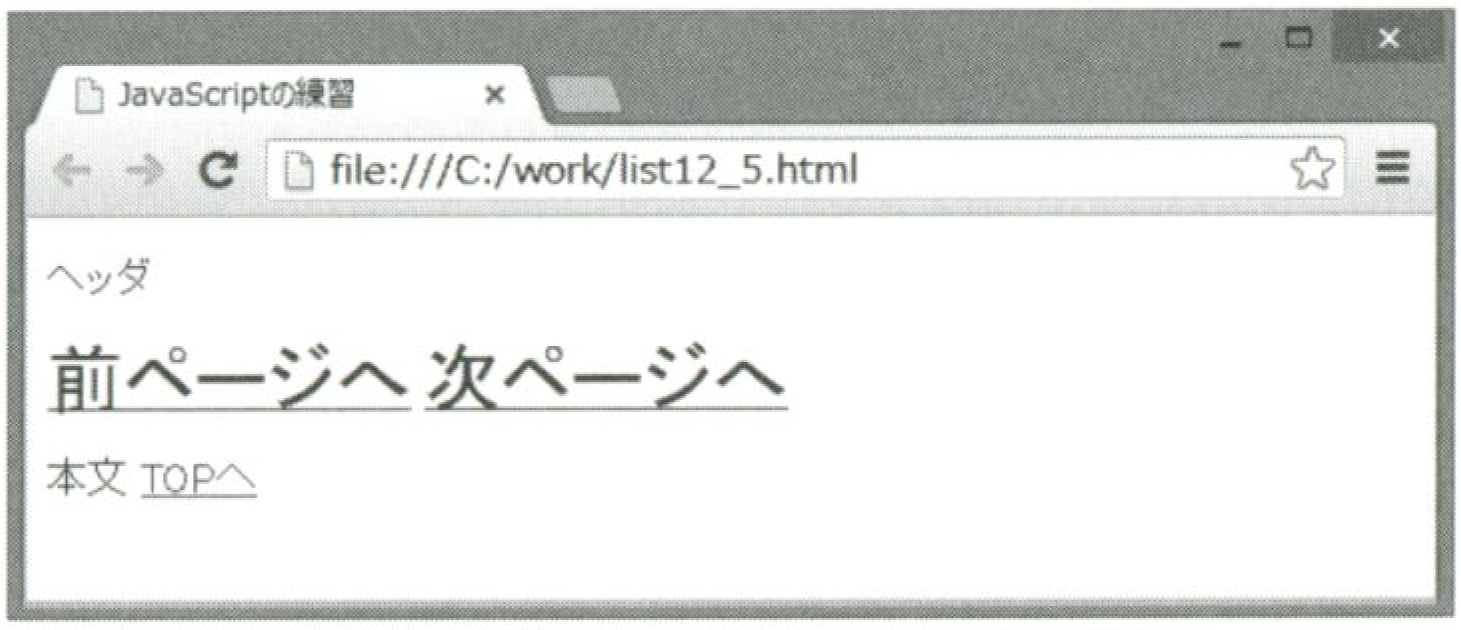
HTML　（<body>要素内のみ抜粋）

```
<p>ヘッダ</p>
<a href="prev.html">前ページへ</a>
<a href="next.html">次ページへ</a>
<p>
  本文
  <a href="index.html">TOPへ</a>
</p>
```

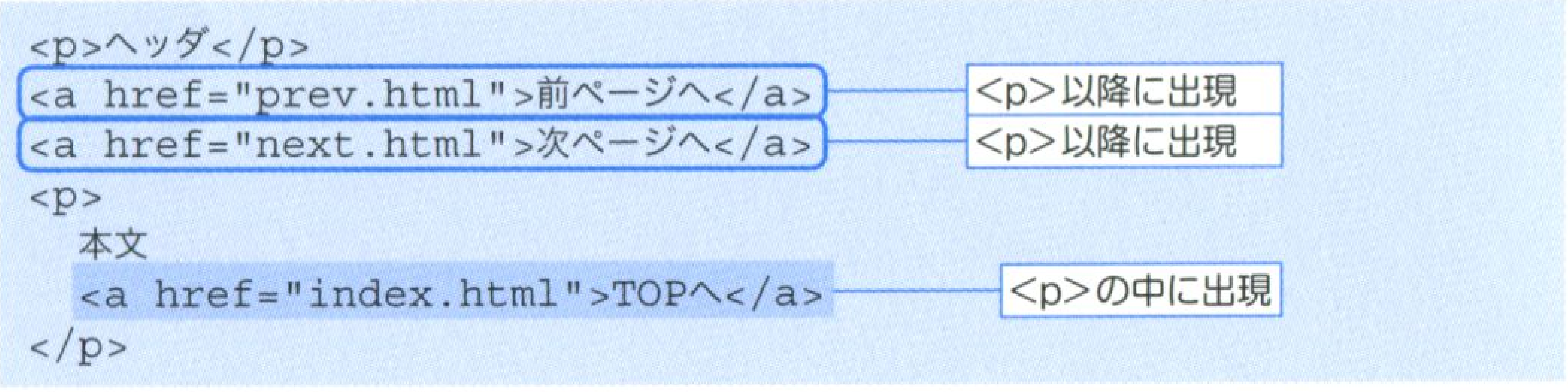
JavaScript

```
$(function() {
    $("p ~ a").css("font-size", "30px");
});
```

■実行結果



リスト12.5は、<p>要素以降に現れる<a>要素すべてに対してスタイルを適用しています。「TOPへ」リンクは<p>要素の兄弟要素ではなく子要素なので、対象外となります。



Point!

▶ jQueryのセレクタは、CSSのセレクタと同じ記述方法で指定することができる。



■ Check Test 2

Q1 以下は、<div>要素の中のすべての<p>要素の文字色を赤に変更するスクリプトです。空欄を埋めて完成させなさい。

```
$("[        ]").css("color", "red");
```

Q2 以下は、<div>要素の中のすべての子要素の文字色を赤に変更するスクリプトです。空欄を埋めて完成させなさい。

```
$("[        ]").css("color", "red");
```

➲解答は巻末に掲載

12 3 要素の抽出

フィルタ

フィルタは、セレクタで取得した要素に対して特定条件による絞り込みを行なう機能です。セレクタだけでは表現できない、より細やかな条件で要素を取得する場合に使います。セレクタで取得した要素をフィルタにかけることで、より複雑な条件による要素の抽出が可能となります。フィルタを利用すれば、リストの背景色を1行おきに変更したり、選択状態になっている要素を強調表示したりといったスクリプトを簡単に記述できます。セレクタで取得した要素をフィルタにかけることで、より複雑な条件による要素の抽出が可能となります。

フィルタは、セレクタの後ろに続けて記述します。

文法 フィルタ

```
$("セレクタフィルタ").メソッド();
```

表12.4は、主なフィルタです。以降で使い方を見ていきましょう。

表 12.4 フィルタ一覧

フィルタ	対象
:first	先頭の要素を取得する
:last	末尾の要素を取得する
:even	偶数番目の要素を取得する
:odd	奇数番目の要素を取得する
:contains(文字列)	指定した文字列を含む要素を取得する
:not(要素)	指定した要素を除外する
:eq(インデックス)	指定したインデックスを持つ要素を取得する
:selected	セレクトボックス等の項目の中で、選択状態になっている要素を取得する
:checked	ラジオボタンやチェックボックスの中で、チェックが入っている要素を取得する
[属性名=値]	指定した属性値を持つ要素を取得する

:firstフィルタ、:lastフィルタ

セレクタが示す要素の中で、先頭／末尾の要素を取得します。

リスト12.6　先頭要素の取得

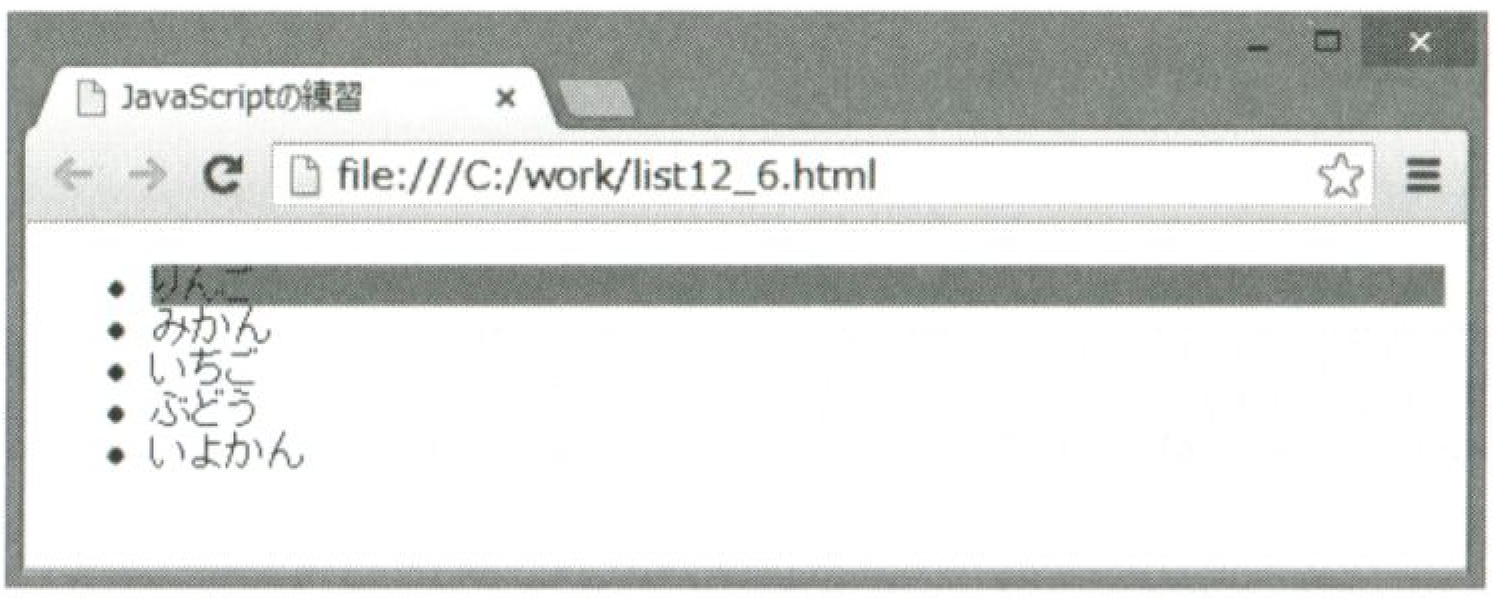
HTML （<body>要素内のみ抜粋）

```
<ul id="fruits">
 <li>りんご</li>
 <li>みかん</li>
 <li>いちご</li>
 <li>ぶどう</li>
 <li>いよかん</li>
</ul>
```

JavaScript

```
$(function() {
    $("#fruits > li:first").css("background-color", "red");
});
```

■実行結果



リスト12.6は、「fruits」というIDを持つ要素の子要素である<li>要素の中で、一番先頭にあたる要素にスタイルを適用しています。結果、「りんご」を内容として持つ<li>要素の背景色のみが変更されています。

```
$("#fruits > li:first").css("background-color", "red");
```

❶ IDが「fruits」の要素を親として持つ<li>要素をすべて取得する（`#fruits > li`）

❷ セレクタで取得した<li>要素のうち先頭の要素を抽出する（`:first`）

要素の絞り込み

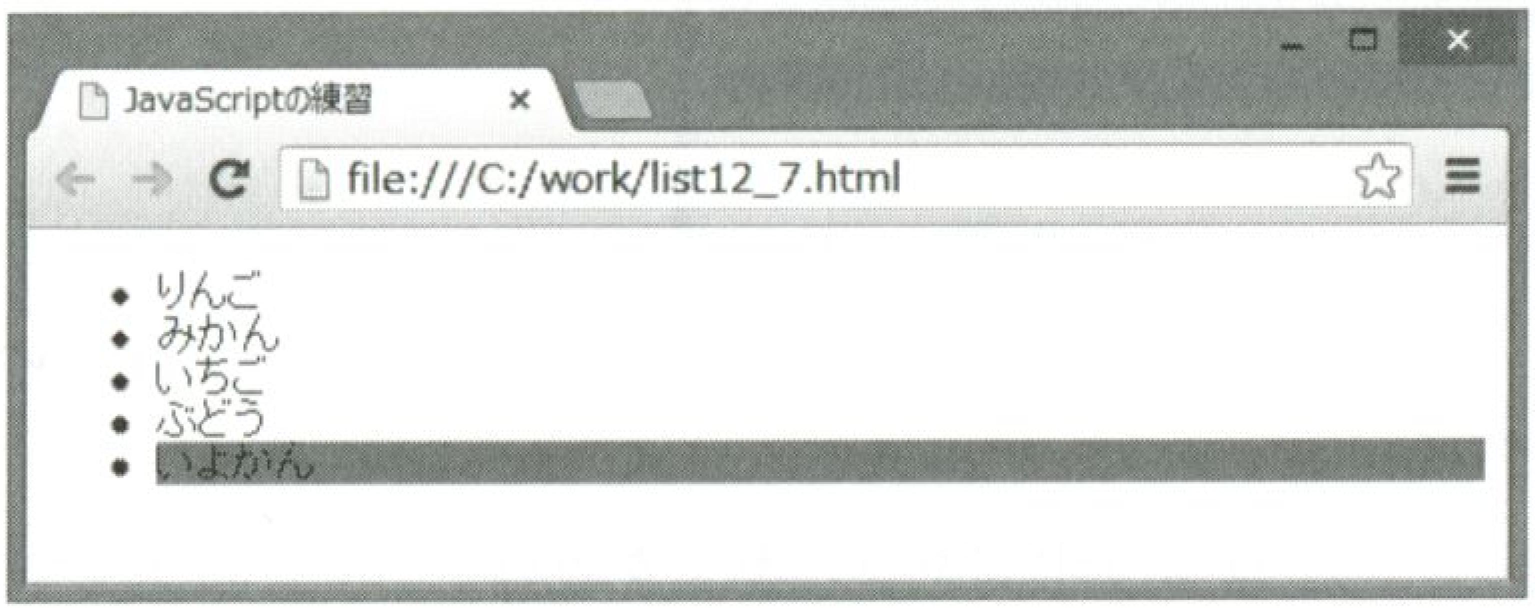
続けて、スクリプトを一部変更してその他のフィルタによる効果も見てみましょう。

リスト12.7では、セレクタで取得した要素のうち末尾の要素に対してスタイルを適用しています。

リスト12.7 末尾要素の取得

```
$(function() {
    $("#fruits > li:last").css("background-color", "red");
});
```

■実行結果



:evenフィルタ、:oddフィルタ

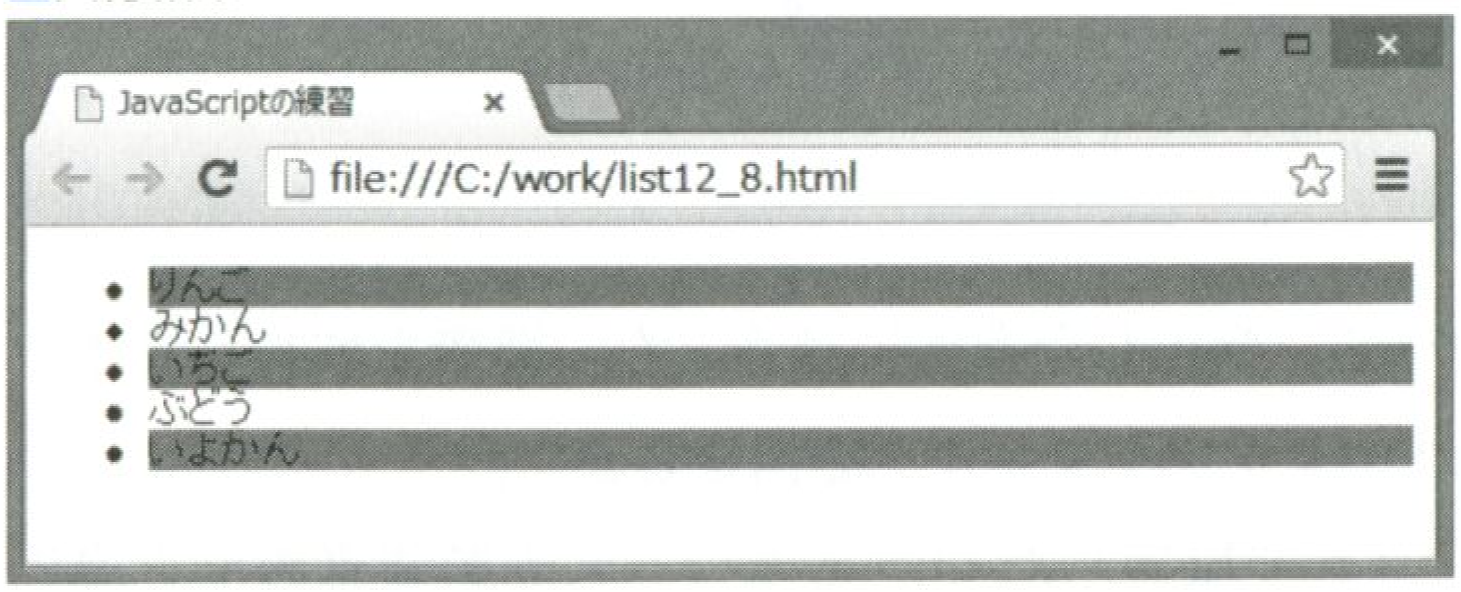
セレクタが示す要素のうち、偶数番目／奇数番目の要素を取得します。

リスト12.8ではセレクタで取得した要素のうち偶数番目の要素にのみスタイルを適用しています。また、リスト12.9では奇数番目の要素を対象としています。先頭の要素を0番目として数え始めることに注意しましょう。

リスト12.8 偶数番目（0、2、4番目）の要素の取得

```
$(function() {
    $("#fruits > li:even").css("background-color", "red");
});
```

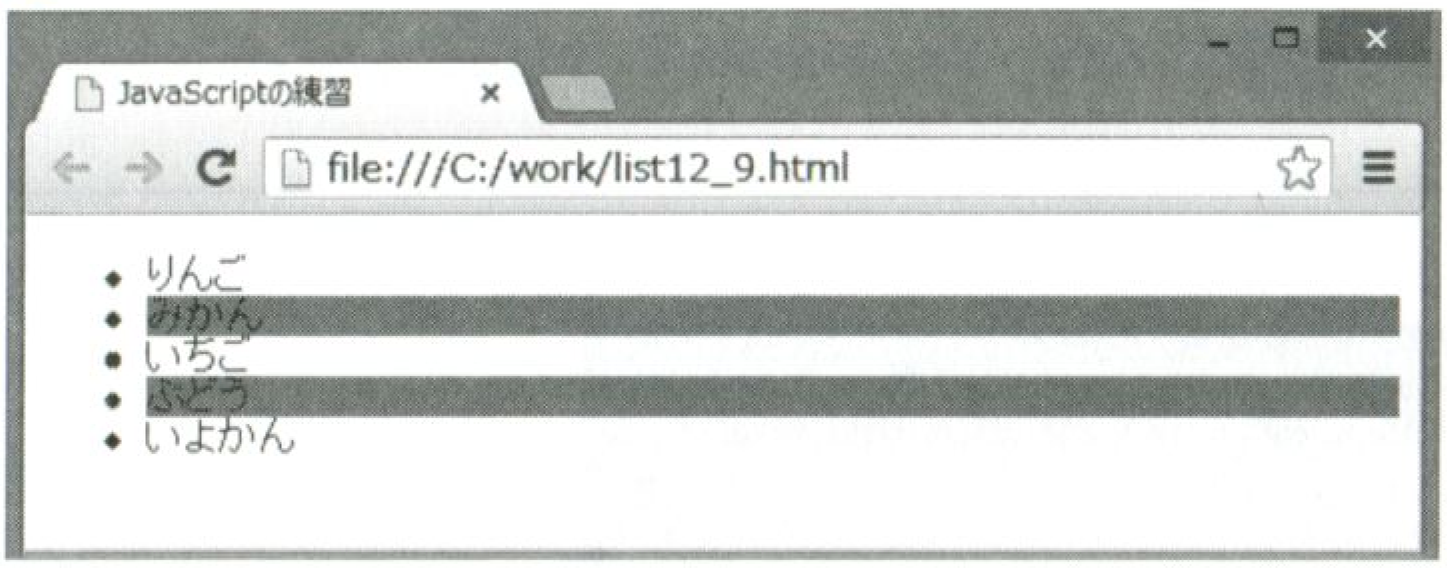
■実行結果



リスト12.9　奇数番目（1、3番目）の要素の取得

```
$(function() {
    $("#fruits > li:odd").css("background-color", "red");
});
```

■実行結果



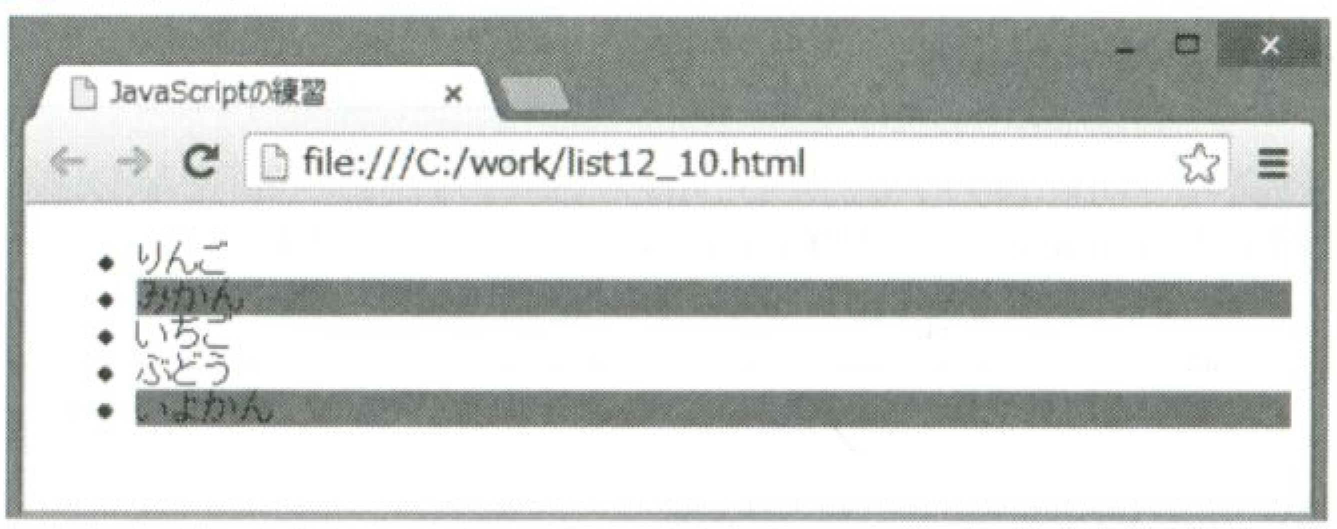
:containsフィルタ

セレクタが示す要素のうち、検索文字列を含む要素を取得します。検索文字列はフィルタの引数として指定します。

リスト12.10　指定文字列を含む要素の取得

```
$(function() {
    $("#fruits > li:contains('かん')").css("background-
color", "red");
});
```

■実行結果



:notフィルタ

セレクタが示す要素のうち、指定した条件に該当しない要素を取得します。引数として除外要素を表すセレクタやフィルタを指定します。

リスト12.11では、セレクタで取得した<li>要素のうち先頭要素のみを除外してスタイルを適用しています。

リスト12.11 除外検索

```
$(function() {
    $("#fruits > li:not(:first)").css("background-color", "red");
});
```

■実行結果



:eqフィルタ

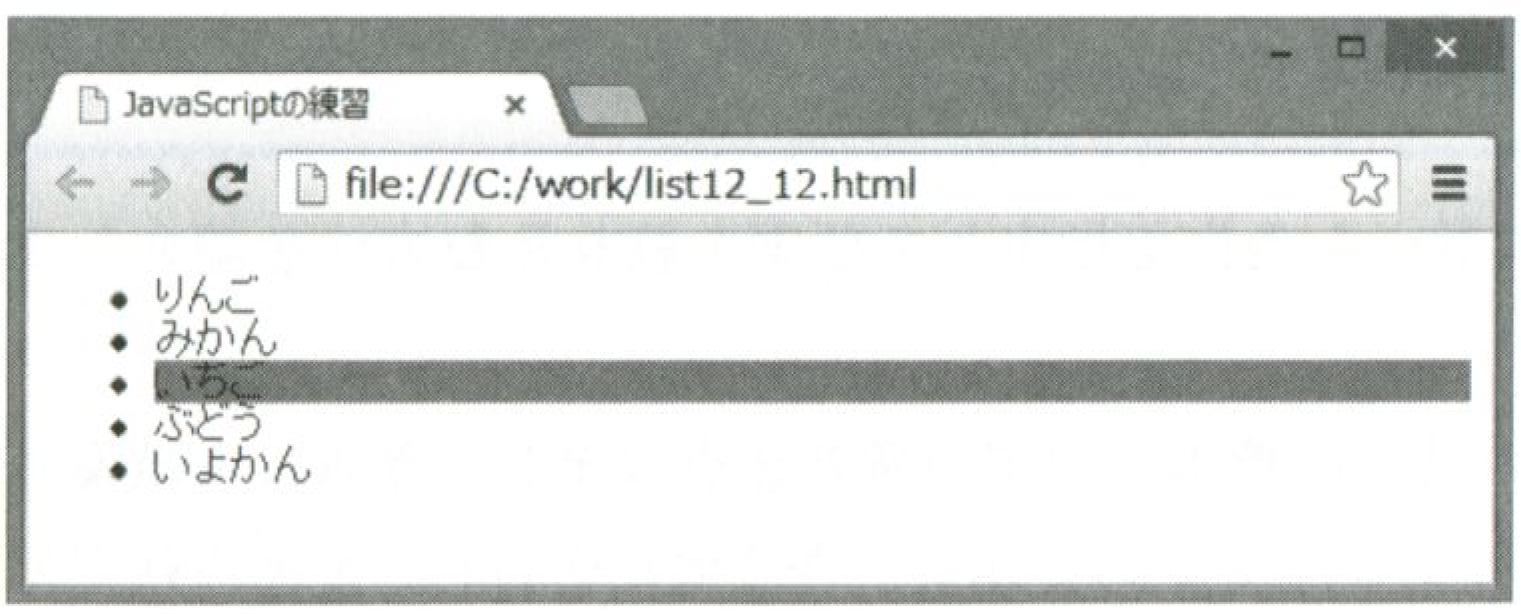
セレクタが示す要素のうち、指定したインデックスを持つ要素を取得します。インデックスは配列と同じように0番目から始まります。

- ●りんご ──→ 0番目
- ●みかん ──→ 1番目
- ●いちご ──→ 2番目
- ●ぶどう ──→ 3番目
- ●いよかん ──→ 4番目

リスト12.12 インデックスを指定した要素の取得

```
$(function() {
    $("#fruits > li:eq(2)").css("background-color", "red");
});
```

■実行結果



◆ 属性フィルタ

セレクタが示す要素のうち、指定した属性名と属性値を持つ要素を取得します。name属性や<input>要素のtype属性などによく使われます。

リスト12.13では、<input>タグのうちname属性に「tel」が設定されている要素にスタイルを設定しています。

リスト12.13　属性フィルタの利用

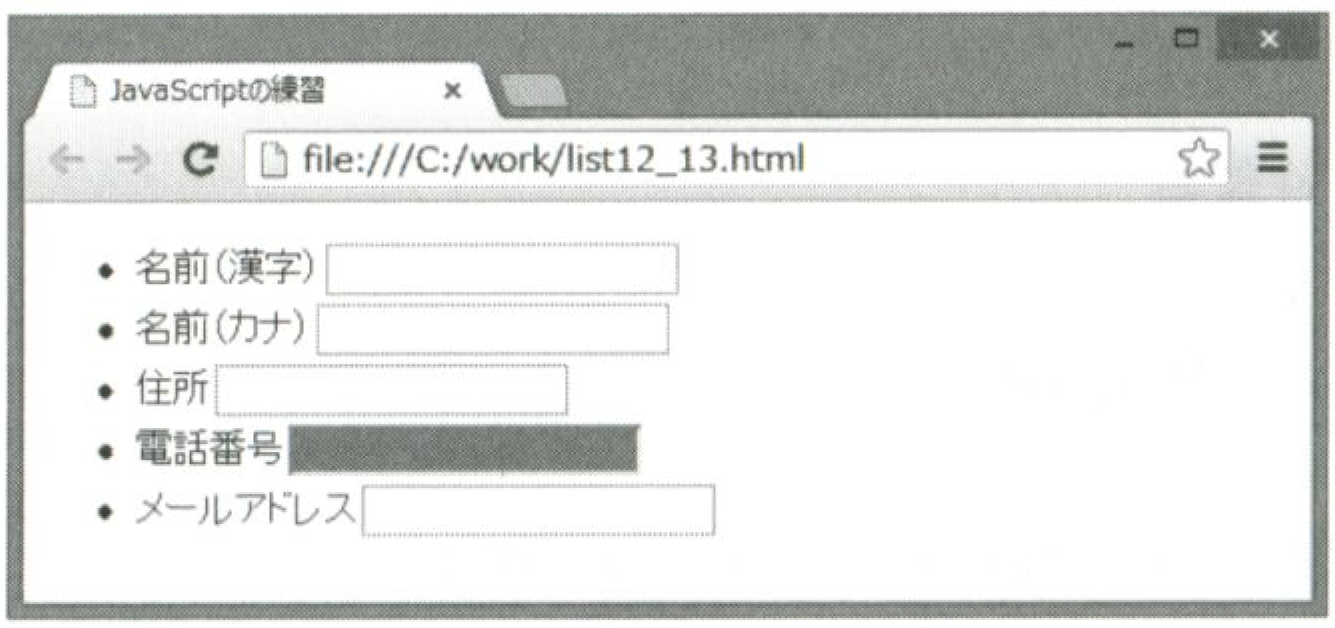
HTML　（<body>要素内のみ抜粋）

```
<ul>
 <li>名前（漢字）<input type="text" name="kanji"></li>
 <li>名前（カナ）<input type="text" name="kana"></li>
 <li>住所<input type="text" name="address"></li>
 <li>電話番号<input type="text" name="tel"></li>
 <li>メールアドレス<input type="text" name="email"></li>
</ul>
```

JavaScript

```
$(function() {
    $("input[name='tel']").css("background-color", "red");
});
```

■実行結果



要素の絞り込み

続いては、ある要素群から条件を指定して要素を絞り込む方法を紹介します。

たとえば、「リストが持つ子要素すべての背景色を変更し、さらに奇数番目の要素は文字色も変更したい」といった場合、今までに学習した記述方法だと次のようなスクリプトになります。

リスト12.14 要素に対して複数回スタイルを設定する

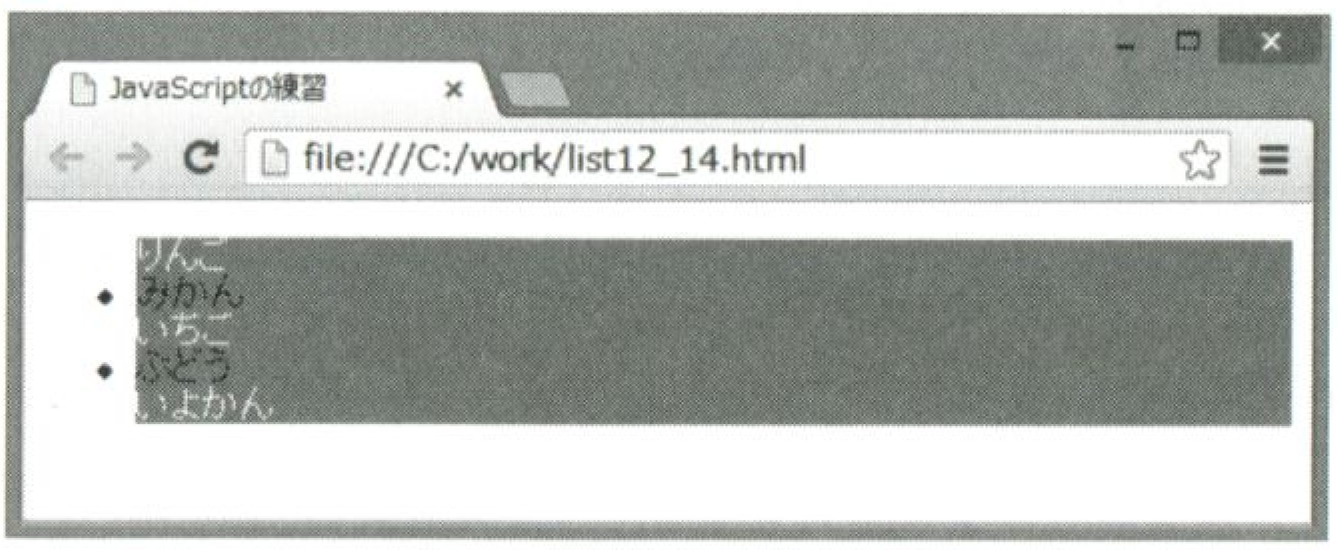
HTML （<body>要素内のみ抜粋）

```
<ul id="fruits">
 <li>りんご</li>
 <li>みかん</li>
 <li>いちご</li>
 <li>ぶどう</li>
 <li>いよかん</li>
</ul>
```

JavaScript

```
$(function() {
    $("#fruits > li").css("background-color", "red");
    $("#fruits > li:even").css("color", "white");
});
```

■実行結果



しかし、この記述方法はあまり好ましくありません。"#fruits > li" に合致する要素を検索する作業が2回発生しているため、その分スクリプトの実行に時間がかかってしまうのです。

```
$("#fruits > li").css("background-color", "red");
         ❶
$("#fruits > li:even").css("color", "white");
         ❷
```

同じ要素を何度も検索しないようにするには、2つの方法があります。

◆ 処理速度の高速化

1つ目は、取得した要素を変数に代入しておく方法です。リスト12.15では、"#fruits > li" に合致した要素群を変数elmに代入しています。以降は変数elmに対してスタイルの適用処理を行なえば良いのです。

今回は変数elmが持つ要素のうち奇数番目の文字色も変更するので、要素を絞り込むためのfilterメソッドを利用しています。

文法 filterメソッド

```
要素群.filter("セレクタやフィルタ");
```

filterメソッドは、要素群からセレクタやフィルタに合致する要素のみを抽出します。

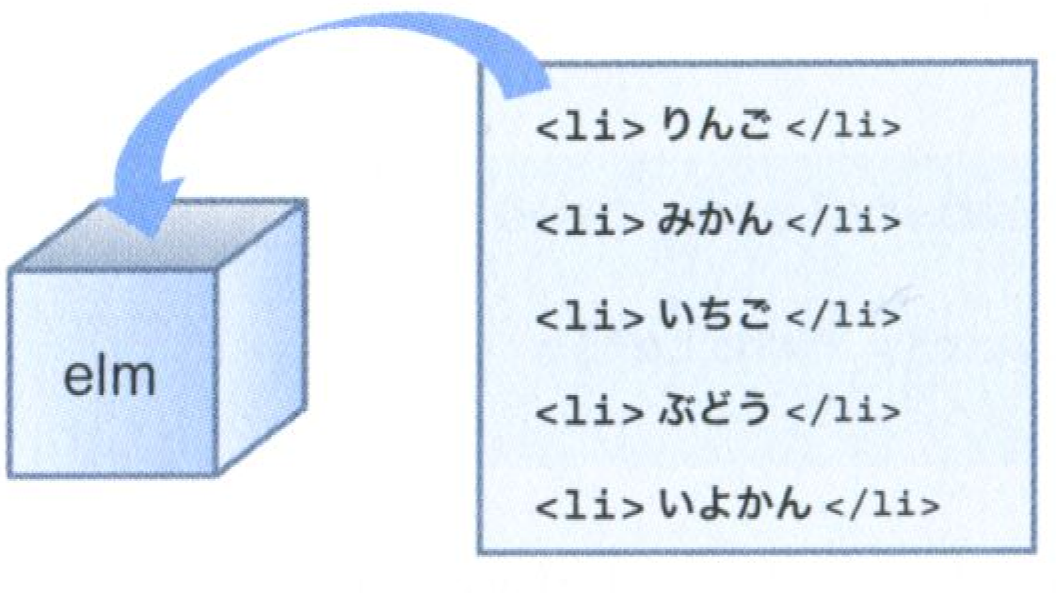
リスト12.15 filterメソッドの利用

```
$(function() {
    var elm = $("#fruits > li");
    elm.css("background-color", "red");
    elm.filter(":even").css("color", "white");
});
```

次のように3段階で処理を行なっています。

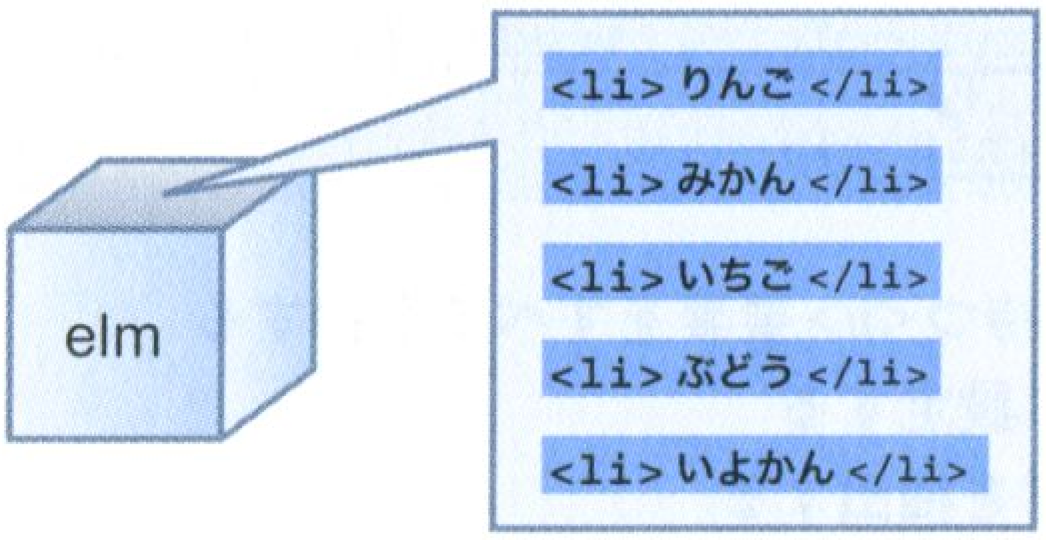
❶取得した要素群を変数elmに代入する

```
var elm = $("#fruits > li");
```



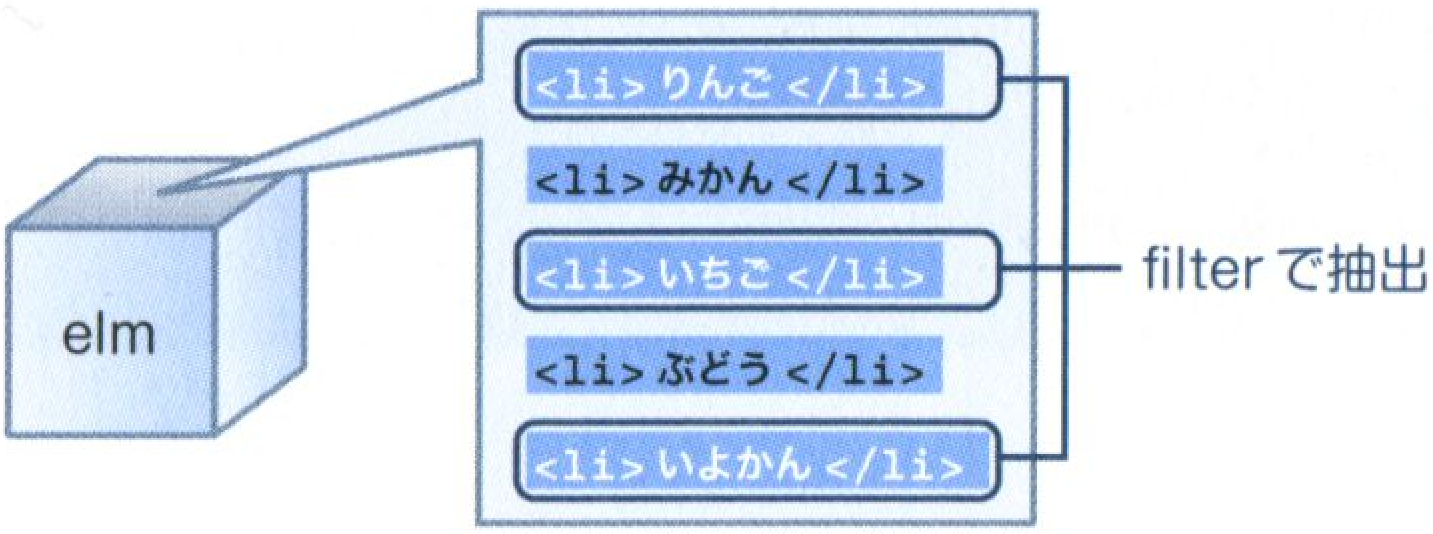
❷変数elmに代入されているすべての要素の背景色を赤に変更する

```
elm.css("background-color", "red");
```



❸変数elmに代入されている要素の中から奇数番目のみを抽出し、文字色を白に変更する

```
elm.filter(":even").css("color", "white");
```



◆ メソッドチェーン

要素の検索回数を抑えるもう1つの方法として、**メソッドチェーン**があります。メソッドチェーンは、メソッドの後ろに続けて別のメソッドをつなげる記法で、要素に対して複数の処理を実行できます。リスト12.16は、前述のスクリプトを書き直したものです。"#fruits > li" に合致する要素群を取得して背景色を設定した後、さらに要素を奇数番目のみに絞り込んで文字色を設定しています。

リスト12.16 メソッドチェーンの利用

```
$(function() {
    $("#fruits > li").css("background-color", "red").↵
filter(":even").css("color", "white");
});
```

このスクリプトは、次のように4つのパートから成り立っています。

```
$("#fruits > li").css("background-color", "red").⏎
       ❶                           ❷
filter(":even").css("color", "white");
       ❸                ❹
```

❶IDが「fruits」の要素を親として持つ<li>要素をすべて取得する
❷❶で取得した要素の背景色を赤に設定する
❸❶で取得した要素群から奇数番目を抽出する
❹❸で抽出した要素の文字色を白に設定する

メソッドチェーンを使うと1行が長くなってしまうので、次のようにドットの前で改行すると見やすくなります。

```
$(function() {
    $("#fruits > li")
        .css("background-color", "red")
        .filter(":even")
        .css("color", "white");
});
```

◆ 子孫要素の絞り込み

filterメソッドによく似た、findというメソッドがあります。この2つは混同しやすいので、サンプルコードを見てその違いを理解しましょう。

文法 findメソッド

```
要素群.find("セレクタやフィルタ");
```


findメソッドは、要素群の子孫要素の中からセレクタやフィルタに合致する要素のみを抽出します。

リスト12.17 filterメソッドとfindメソッド

HTML （<body>要素内のみ抜粋）

```
<div class="item">
 <p>あいうえお</p>
</div>
<div>
 <p class="item">かきくけこ</p>
</div>
```

JavaScript

```
$(function() {
    console.log($("div").filter(".item").html());
    console.log($("div").find(".item").html());
});
```

■実行結果

```
<p>あいうえお</p>
かきくけこ
```

filterメソッドは、<div>要素のうち「item」というCSSクラスを持つ要素を取得します。

```
$("div").filter(".item").html();
```

```
<div class="item">
 <p>あいうえお</p>
</div>
<div>
 <p class="item">かきくけこ</p>
</div>
```

それに対してfindメソッドは、<div>要素の子孫要素の中から「item」というCSSクラスを持つ要素を取得します。

```
$("div").find(".item").html();
```

```
<div class="item">
 <p>あいうえお</p>
</div>
<div>
 <p class="item">かきくけこ</p>
</div>
```

htmlメソッドは、開始タグと終了タグの間のHTMLを返します。innerHTMLプロパティが表わす値と同じです。詳しくは次節で解説します。

Point!

▶ フィルタやfilter／findメソッドを使うと、セレクタで取得した要素を絞り込むことができる。



■Check Test 3

Q1 以下は、<div>要素の子要素のうち、先頭要素の文字色を赤に変更するスクリプトです。空欄を埋めて完成させてください。

```
$("div > *[      ]").css("color", "red");
```

Q2 以下は、<input>要素のうち「hoge」というname属性値を持つ要素の文字色を赤に変更するスクリプトです。空欄を埋めて完成させてください。

```
$("input[        ]").css("color", "red");
```

➲解答は巻末に掲載

12 4 内容と属性の操作

内容と属性の変更

要素の取得方法を理解したところで、続いては要素に対して属性値やイベントハンドラ、CSSクラスを設定する各種メソッドについて学びます。

内容や属性を変更することで、条件に応じて表示されるメッセージを変更したり、画像やCSSクラスの切り替えなどを行なうことができます。

◆ htmlメソッド

要素に含まれる内容を操作します。引数を指定した場合は内容を上書きし、引数を省略した場合は現在の内容を返します。

文法 内容の変更

```
要素.html(値);
```

対象の要素が複数件ある場合、すべての要素の内容を書き換えます。

文法 内容の取得

```
要素.html();
```

対象の要素が複数件ある場合、一番先頭の要素の内容を取得します。

リスト12.18 内容の操作

HTML （<body>要素内のみ抜粋）

```
<p id="msg1">おはよう</p>
<p id="msg2">こんにちは</p>
```

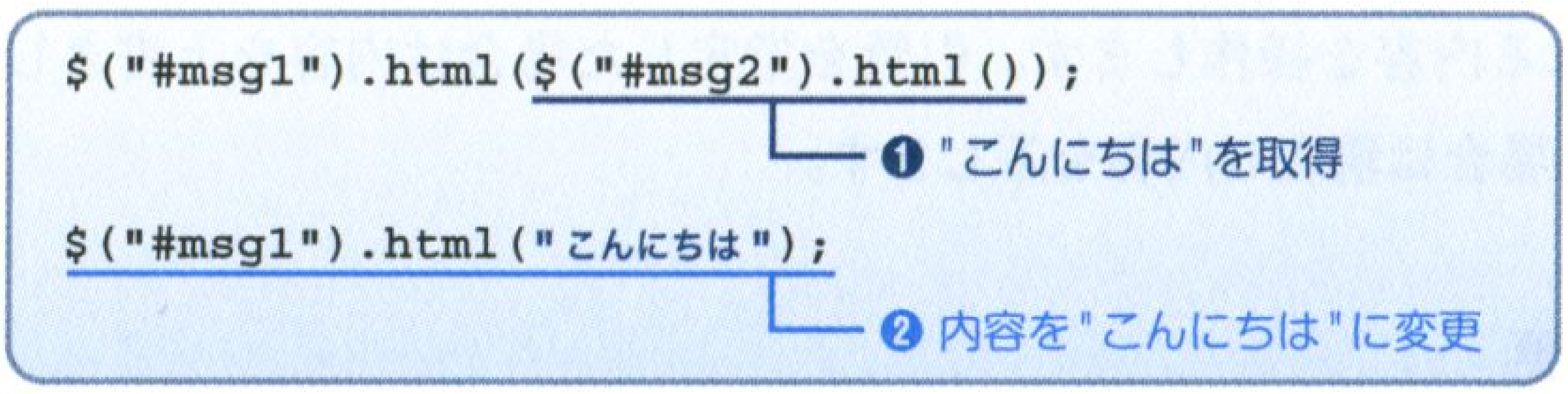
JavaScript

```
$(function() {
    $("#msg1").html($("#msg2").html());
});
```

■実行結果



リスト12.18では、まず「msg2」というIDを持つ要素の内容を取得し、それを「msg1」というIDを持つ要素の内容として設定しています。



処理イメージ

◆ valメソッド

value属性値を操作します。htmlメソッドと同様に、引数を指定した場合はvalue属性値を上書きし、引数を省略した場合は現在のvalue属性値を返します。

文法 value属性値の変更

```
要素.val(値);
```

対象の要素が複数件ある場合、すべての要素のvalue属性値を書き換えます。

文法 value属性値の取得

```
要素.val();
```

対象の要素が複数件ある場合、一番先頭の要素のvalue属性値を取得します。

リスト12.19 value属性値の操作

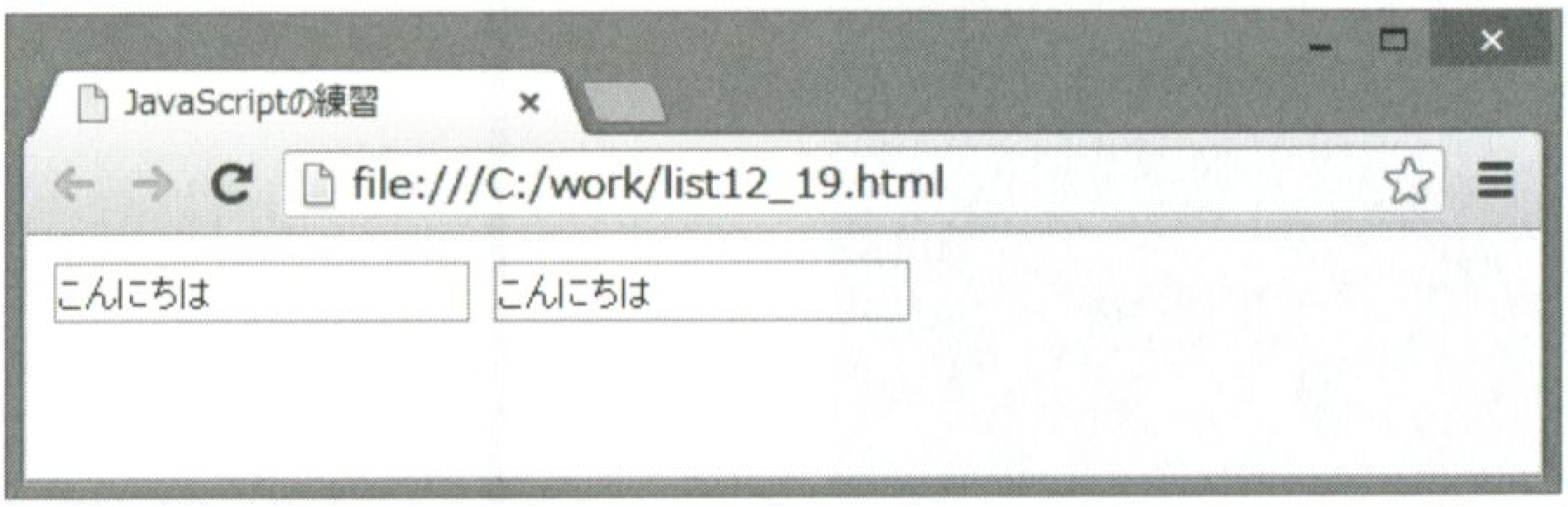
HTML (<body>要素内のみ抜粋)

```
<input type="text" id="msg1" value="おはよう">
<input type="text" id="msg2" value="こんにちは">
```

JavaScript

```
$(function() {
    $("#msg1").val($("#msg2").val());
});
```

■実行結果



◆ attrメソッド

value以外の属性値を変更します。第2引数を指定した場合は属性値を上書きし、省略した場合は現在の属性値を返します。

文法 属性値の変更

```
要素.attr("属性名", 値);
```

対象の要素が複数件ある場合、すべての要素の属性値を書き換えます。

文法 属性値の取得

```
要素.attr("属性名");
```

対象の要素が複数件ある場合、一番先頭の要素の属性値を取得します。

リスト12.20　その他の属性の変更

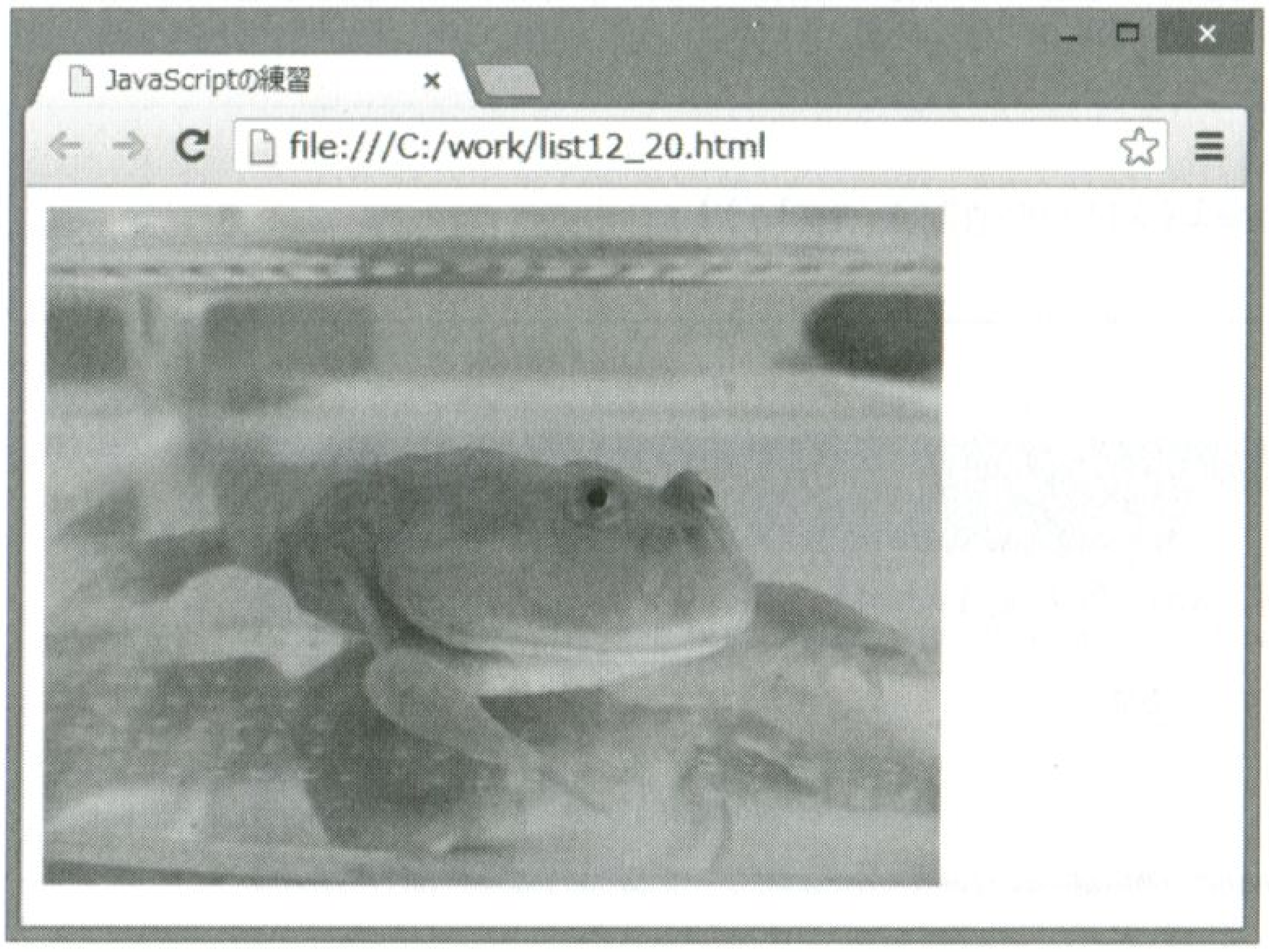
HTML　（<body>要素内のみ抜粋）

```
<img>
```

JavaScript

```
$(function() {
    $("img").attr("src", "images/frog1.jpg");
});
```

■実行結果



リスト12.20を実行すると、空の<img>要素に対してsrc属性が設定され、次のように記述した場合と同じ状態になります。

```
<img src="images/frog1.jpg">
```

◇ removeAttrメソッド

指定した属性をすべての要素から削除します。

文法 属性の削除

```
要素.removeAttr("属性名");
```

属性の削除は、次のようにattrメソッドで属性の値に空文字を設定することでも行なうことができます。

```
$("img").attr("src", "");
```

しかし、selected、changed、enabled、disabledのような、値の省略が可能な属性はremoveAttrメソッドでなければ削除できません。

リスト12.21　その他の属性の変更

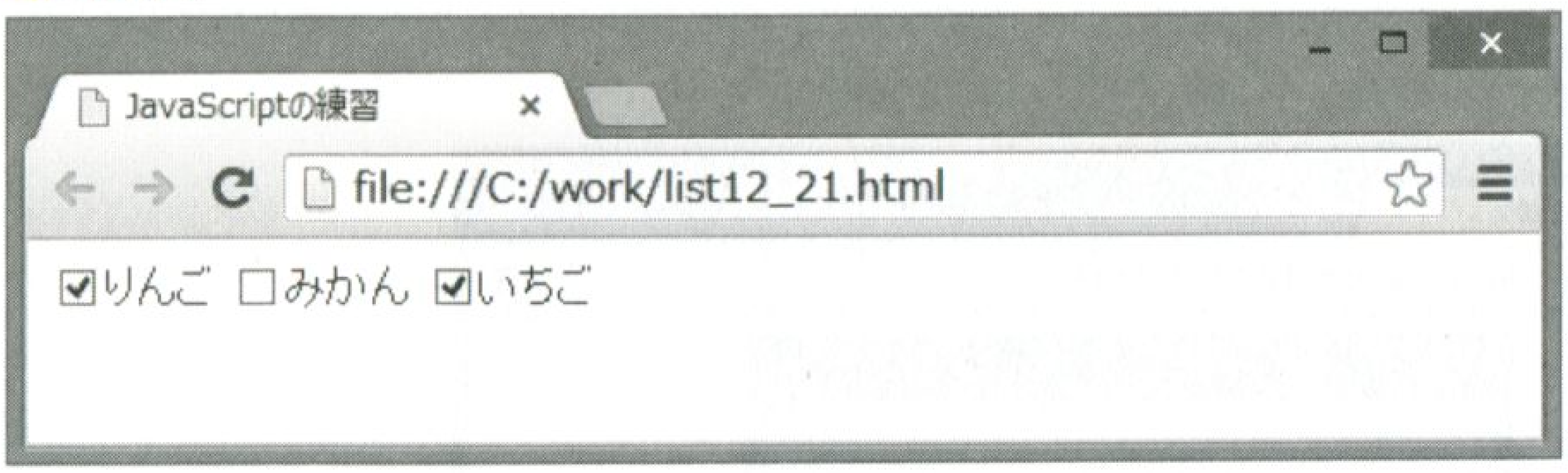
HTML　(<body>要素内のみ抜粋)

```
<input type="checkbox" name="fruits" value="apple"
checked>りんご
<input type="checkbox" name="fruits" value="orange"
checked>みかん
<input type="checkbox" name="fruits" value="strawberry"
checked>いちご
```

JavaScript

```
$(function() {
    $("input[value='orange']").removeAttr("checked");
});
```

■実行結果



イベントハンドラの設定

第10章で学習したDOM要素に対するイベントハンドラの設定をjQueryで行なうには、onメソッドとoffメソッドを利用します。

◆ onメソッド

要素に対してイベントハンドラを登録します。第1引数のイベント名には「on」は付けません。複数のイベント名をスペース区切りで指定すれば、それぞれのイベントに対して同じ処理を設定することができます。第2引数は関数に渡す引数で、オブジェクト型のデータにします。関数が引数を受け取らない場合は省略可能です。第3引数にはイベント発生時に実行する関数を指定します。

文法 onメソッド

```
要素.on("イベント名", 関数に渡す引数, 関数名または関数式);
```

リスト12.22 引数のない関数をイベントハンドラに登録する

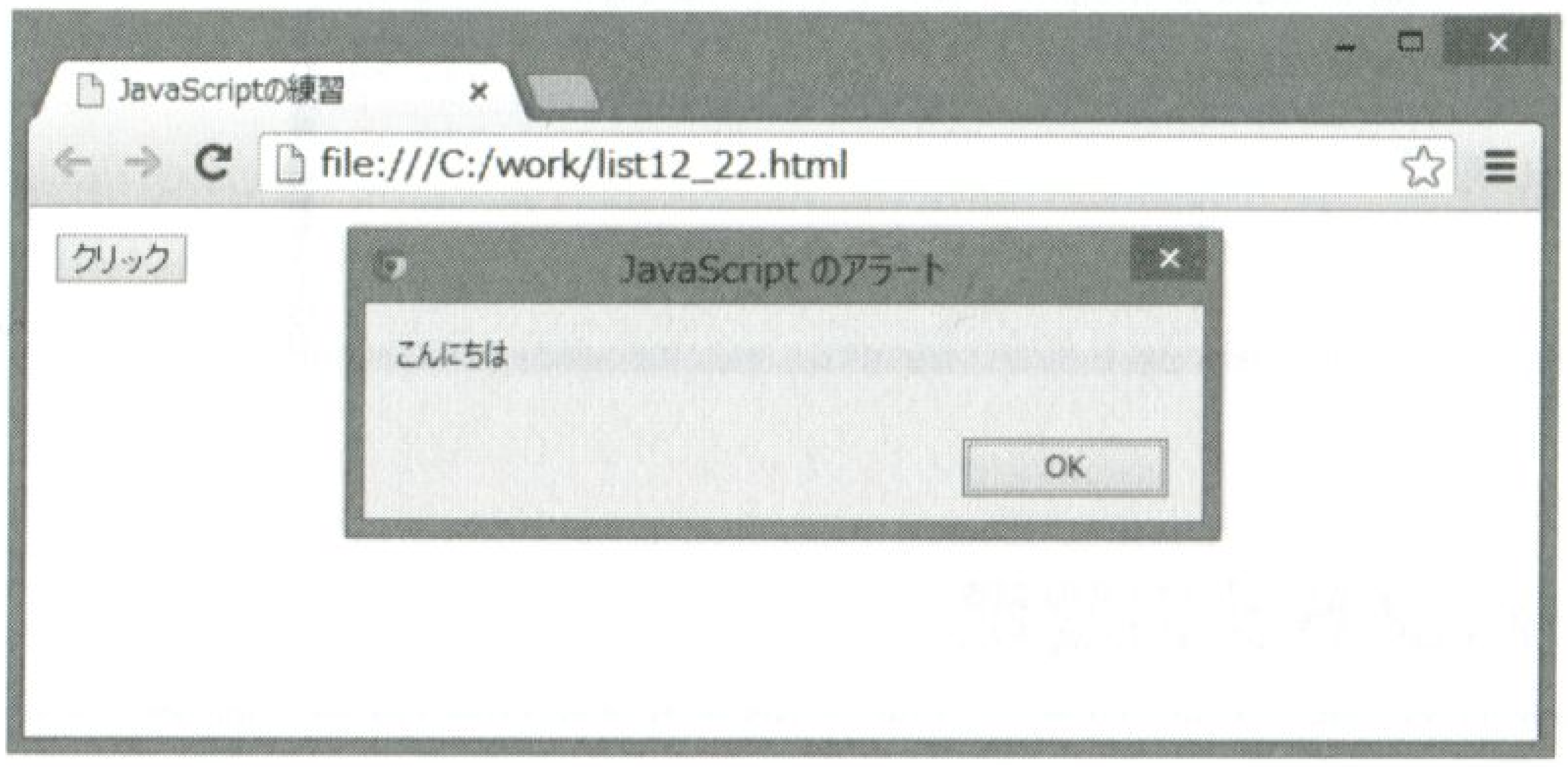
HTML （<body>要素内のみ抜粋）

```
<button id="btn">クリック</button>
```

JavaScript

```
$(function() {
    $("#btn").on("click", sayHello);
});
function sayHello() {
    alert("こんにちは");
}
```

■実行結果



リスト12.22では、ボタンのclickイベント時の処理としてsayHello関数を登録しています。onメソッドが実行されると、<button>タグに対して次のようにイベントハンドラを設定した場合と同じ状態になります。

```
<button id="btn" onclick="sayHello()">クリック</button>
```

sayHello関数を他の用途で利用することがないのであれば、次のように匿名関数としてonメソッドの引数に直接指定してもかまいません。

```
$("#btn").on("click", function(){
    alert("こんにちは");
});
```

イベントハンドラに登録する関数が引数を受け取る場合は、onメソッドの第2引数に関数へ渡す値を設定します。

リスト12.23 引数のある関数をイベントハンドラに登録する

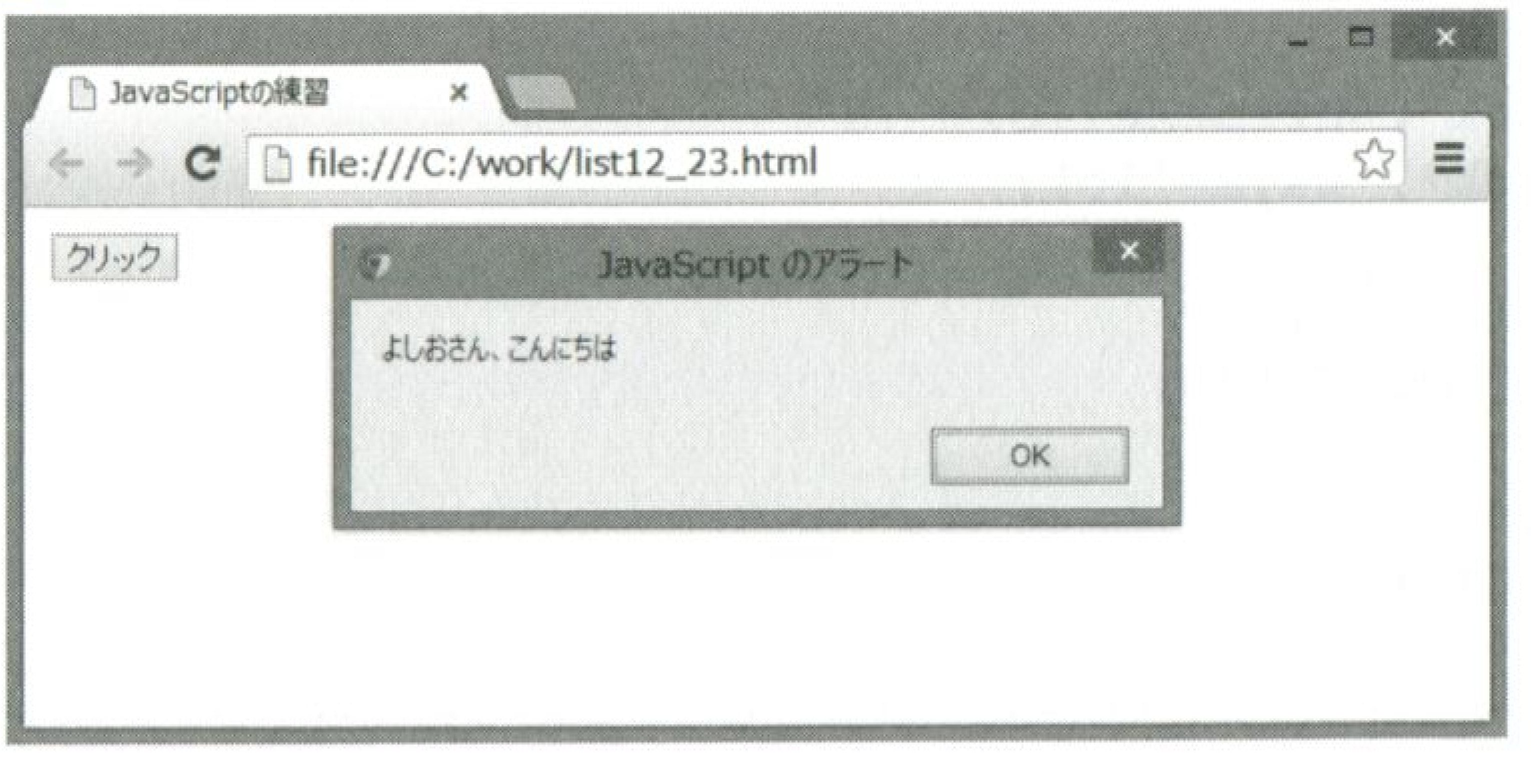
HTML (<body>要素内のみ抜粋)

```
<button id="btn">クリック</button>
```

JavaScript

```
$(function() {
    $("#btn").on("click", { name: "よしお" }, sayHello);
});
function sayHello(event) {
    alert(event.data.name + "さん、こんにちは");
}
```

■実行結果



引数はオブジェクト型（連想配列）にして渡します。on関数を使って引数を渡した場合、普通に関数を呼び出したときとは異なる引数の渡し方になります。引数として渡されたオブジェクトは、関数側で受け取った変数のdataプロ

パティの中に代入されています。

```
$(function() {
    $("#btn").on("click", { name: "よしお" }, sayHello);
});
function sayHello(event) {
    alert(event.data.name + "さん、こんにちは");
}
```

event.dataの中に代入される

これを図にすると、次のようなイメージです。



実行イメージ

なお、onメソッドはイベントハンドラの上書きではなく追加なので、同じイベントに対して複数の関数を登録することも可能です。

リスト12.24　1つのイベントに複数の関数を設定する

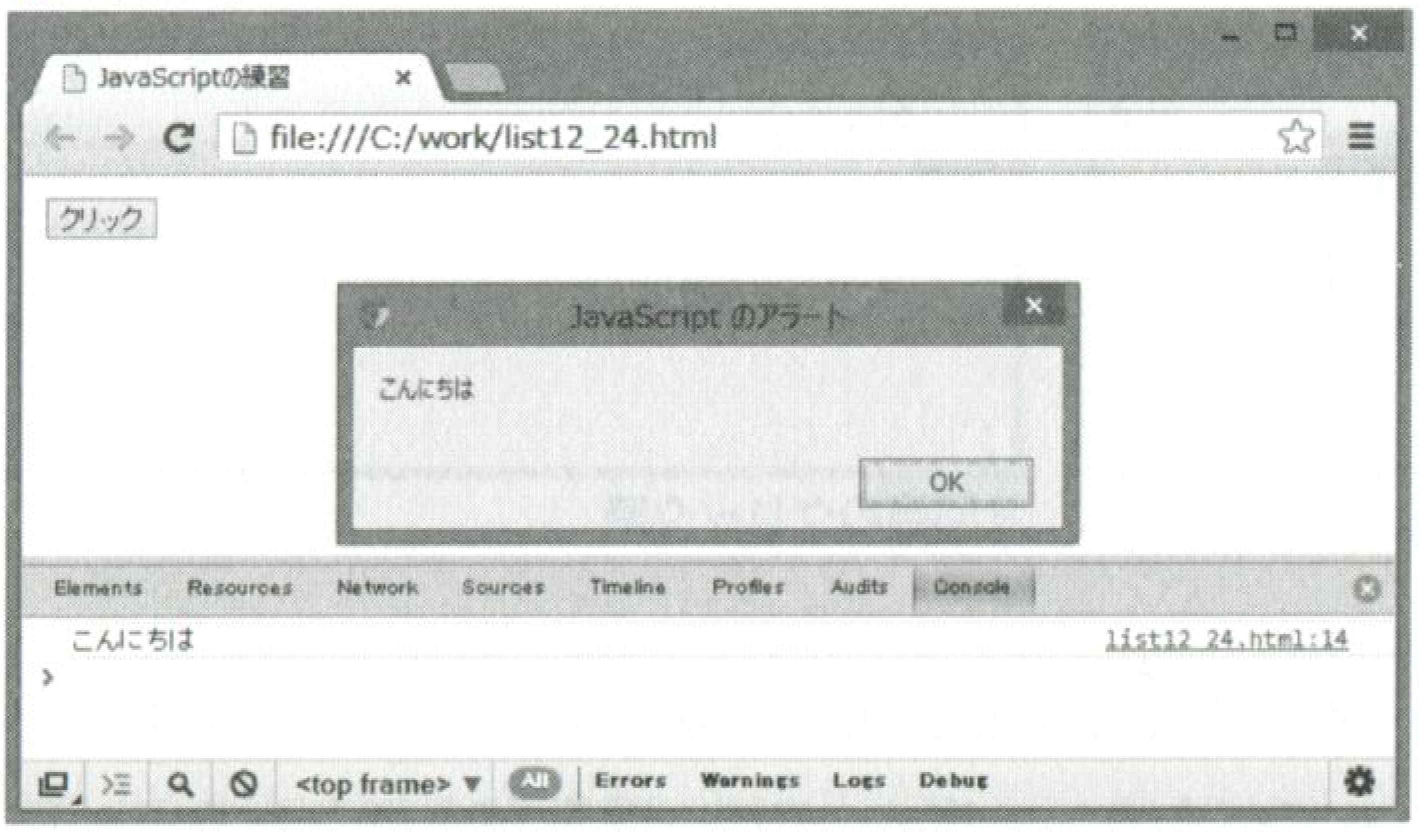
HTML　(<body>要素内のみ抜粋)

```
<button id="btn">クリック</button>
```

JavaScript

```
$(function() {
    $("#btn")
        .on("click", printHello)
        .on("click", sayHello);
});
function printHello() {
    console.log("こんにちは");
}
function sayHello() {
    alert("こんにちは");
}
```

■実行結果



リスト12.24では、clickイベントに対してprintHello関数とsayHello関数を登録しています。click時には2つの関数が登録した順に実行されます。

◇ offメソッド

要素に関連付けられたイベントハンドラを削除します。引数をすべて省略した場合、click、focusなどの全イベントハンドラが削除されます。第2引数のみを省略した場合、指定したイベントに設定されたすべての関数が削除されます。

文法 offメソッド

```
要素.off("イベント名", 関数名);
```

リスト12.25 イベントハンドラを削除する

HTML （<body>要素内のみ抜粋）

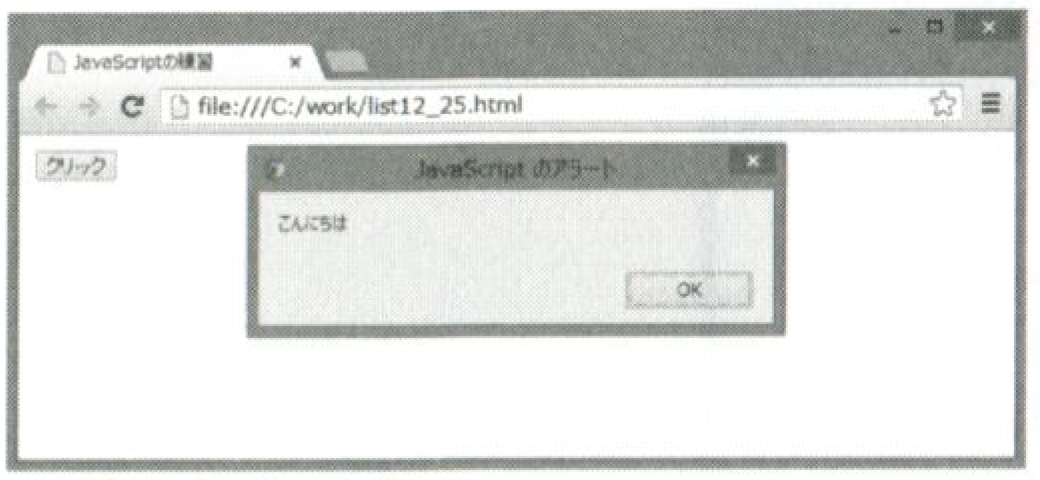
```
<button id="btn">クリック</button>
```

JavaScript

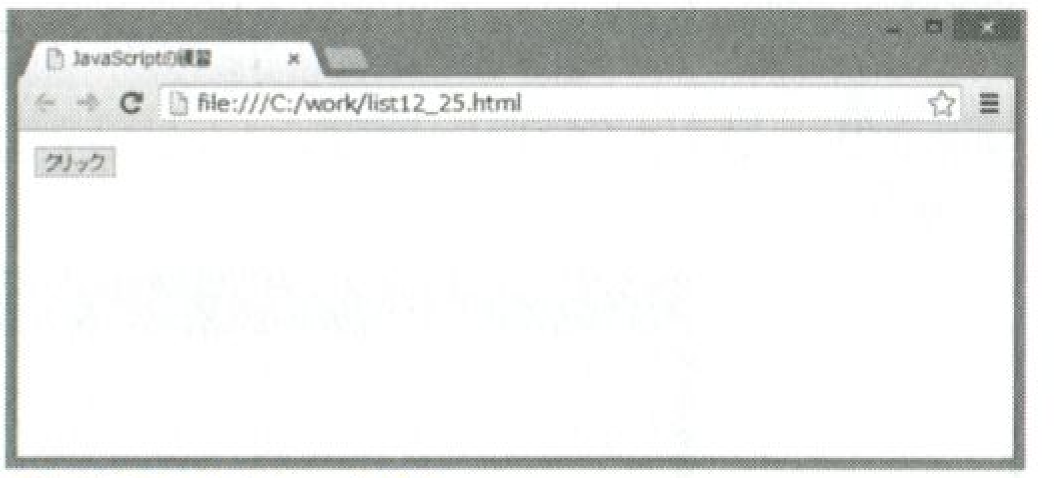
```
$(function() {
    $("#btn").on("click", sayHello);
});
function sayHello() {
    alert("こんにちは");
    $("#btn").off("click", sayHello);
}
```

12

■実行結果



1回目のクリック時



2回目のクリック時

リスト12.25では、クリック時に実行するsayHello関数の中で、イベントハンドラを削除する処理を行なっています。一度sayHello関数が実行されると、関連付けられた処理はなくなるので、2回目以降のクリック時にはなにも起こりません。

実行イメージ

```
$(function() {
    ❶ボタンのclickイベント時の処理としてsayHello関数を登録
    $("#btn").on("click", sayHello);
});
❷ボタンがclickされるとsayHello関数が実行される
function sayHello() {
    alert("こんにちは");
    $("#btn").off("click", sayHello);   ❸ボタンのclickイベントから
                                          sayHello関数を削除
}
```

◇ イベント発生元の特定

イベントの発生元要素を表わすthisキーワードは、jQueryの場合、次のように記述します。

```
$(this)
```

リスト12.25は2回目以降のボタンクリック時にはなにも処理を行なわないスクリプトでしたので、1回目のクリックが行なわれた際にボタンを無効化する

処理を追加してみましょう。

リスト12.26　jQueryによるイベント発生元の取得

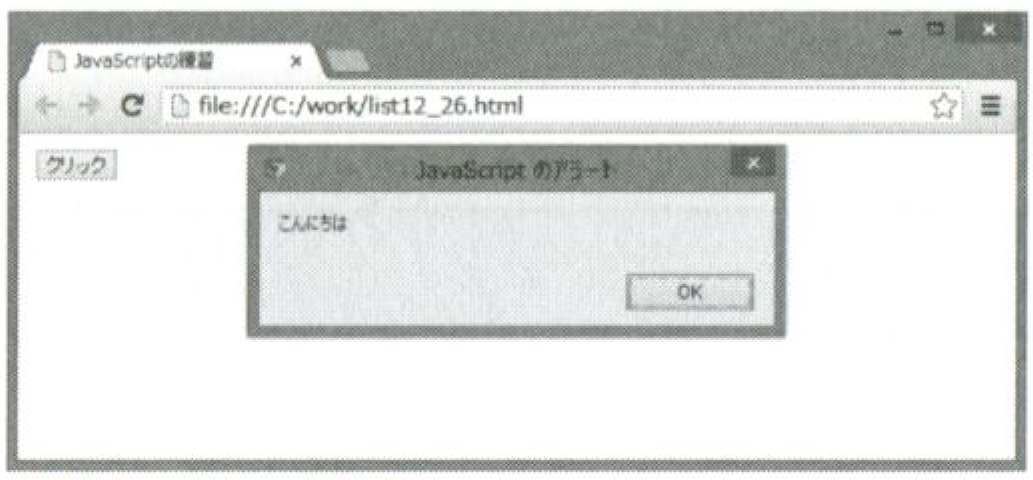
HTML　（<body>要素内のみ抜粋）

```
<button id="btn">クリック</button>
```

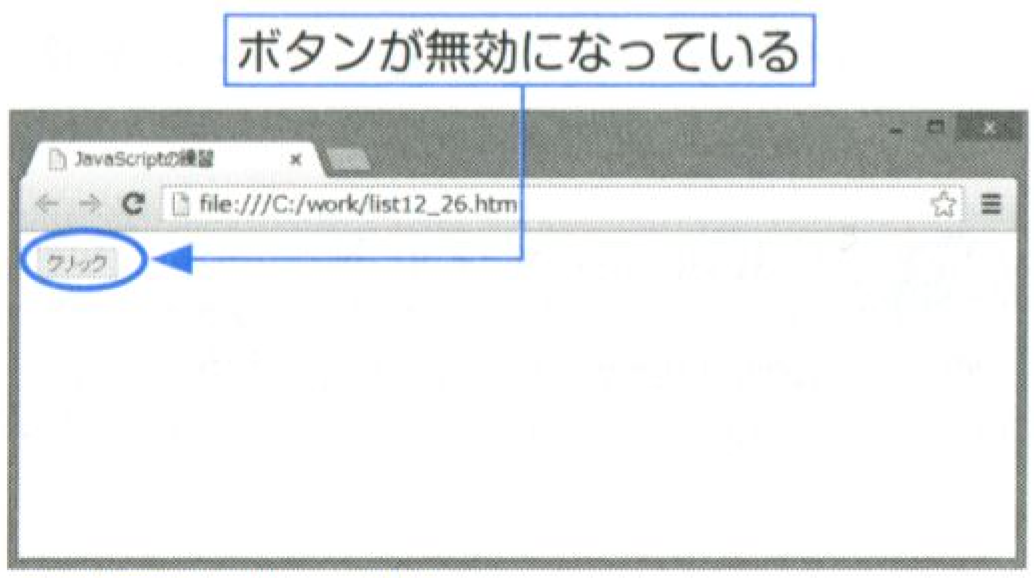
JavaScript

```
$(function() {
    $("#btn").on("click", sayHello);
});
function sayHello() {
    alert("こんにちは");
    $("#btn").off("click", sayHello);
    $(this).attr("disabled", "disabled");
}
```

■実行結果



1回目のクリック時



2回目のクリック時

リスト12.26では、ボタンクリック時の処理の中でイベントの発生元であるボタンにdisabled属性を設定しています。disabled属性は属性名、属性値ともに"disabled"を指定します。

```
$(this).attr("disabled", "disabled");
```

$(this)：クリックイベントの発生元であるボタンを表わす

これにより、1度クリックが行なわれたボタンは無効化された状態で表示されます。

CSSクラスの変更

ここまで、スタイルを変更する処理はcssメソッドで行なっていました。しかし、実際のWeb制作現場では1つの要素に対して複数のスタイルを設定するため、cssメソッドで1つ1つCSSプロパティを指定していると非常に長いコードになってしまいます。ですので、あらかじめCSSプロパティ群をまとめて1つのCSSクラスとして用意しておき、要素に対してCSSクラスを付け替える方法を用いるのが現実的です。

addClassメソッド

要素に対してCSSクラスを追加します。上書きではないので、元々設定されていたCSSクラスが削除されるわけではありません。複数のCSSクラスをスペース区切りで指定することもできます。

文法 addClassメソッド

```
要素.addClass("CSSクラス名");
```

リスト12.27 CSSの追加

HTML （<body>要素内のみ抜粋）

```
<p id="msg" class="large">こんにちは</p>
```

JavaScript

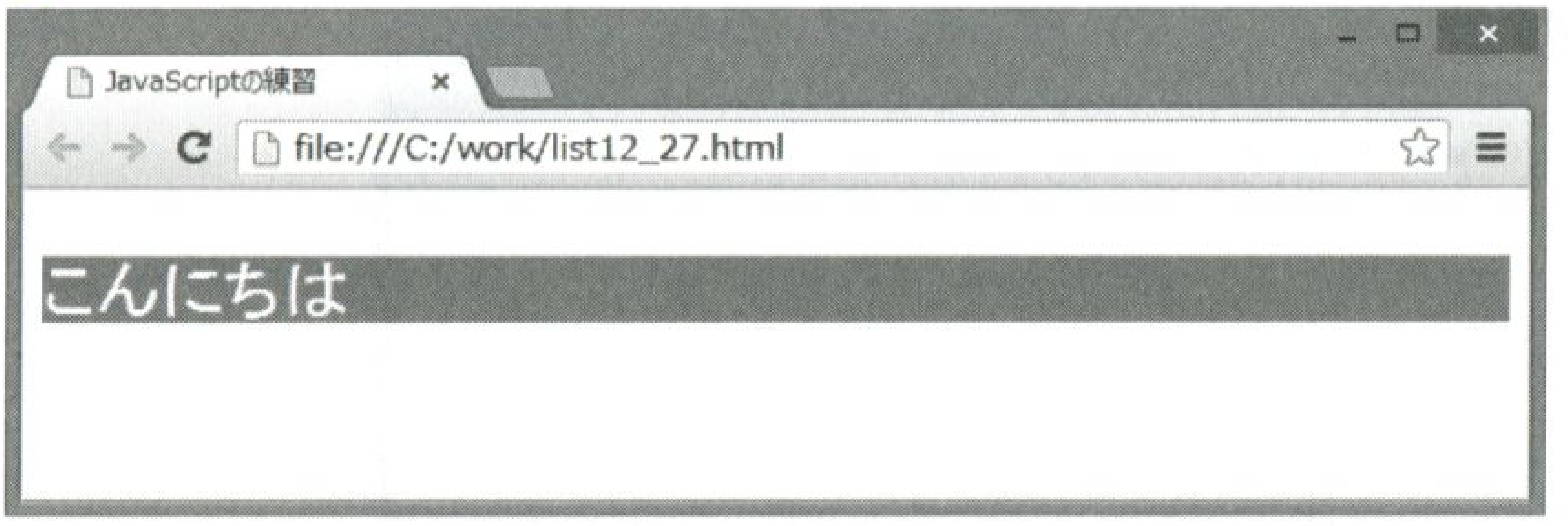
```
$(function() {
    $("#msg").addClass("highlight");
});
```

CSS

```
.large {
    font-size: 30px;
}
.highlight {
    color: white;
    background-color: red;
}
```

12

■実行結果



リスト12.27では、対象の<p>要素に対して「highlight」という名前のCSSクラスを追加しています。addClassメソッドが実行されると、次のように<p>タグのclass属性を記述した場合と同じ状態になります。

```
<p id="msg" class="large highlight">こんにちは</p>
```

◆ removeClassメソッド

要素からCSSクラスを削除します。複数のCSSクラスをスペース区切りで指定することもできます。

文法 removeClassメソッド

```
要素.removeClass("CSSクラス名");
```

リスト12.28　CSSの削除

HTML　(<body>要素内のみ抜粋)

```
<p id="msg" class="large">こんにちは</p>
```

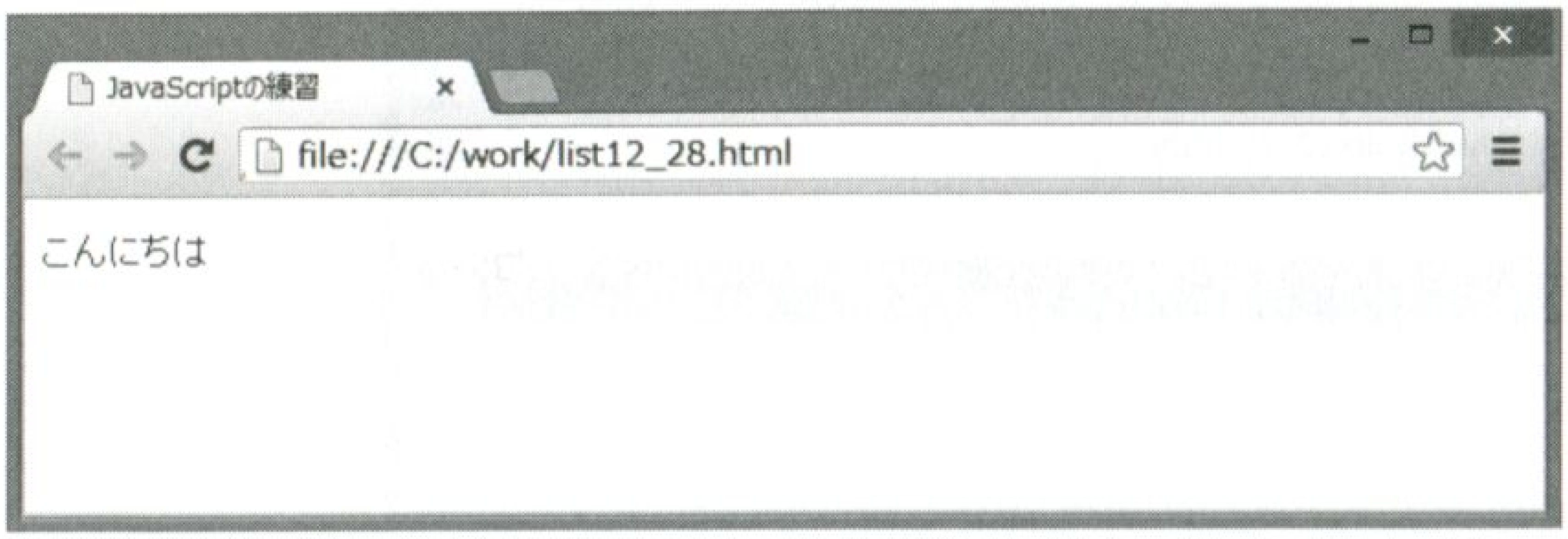
JavaScript

```
$(function() {
    $("#msg").removeClass("large");
});
```

CSS

```
.large {
    font-size: 30px;
}
```

■実行結果



リスト12.28では、対象の<p>要素から「large」という名前のCSSクラスを削除しています。実行後はCSSクラスが1つも適用されていない状態となるため、スタイル未設定の要素が表示されます。

◆ toggleClassメソッド

CSSクラスの追加と削除を交互に繰り返します。クリックされるたびにスタイルが付いたり外れたりする処理などに利用します。複数のCSSクラスをスペース区切りで指定することも可能です。

文法 toggleClassメソッド

```
要素. toggleClass("CSSクラス名");
```

リスト12.29 表示エリアの切り替え

HTML （<body>要素内のみ抜粋）

```
<input type="radio" name="target" value="i" checked>一般用
<input type="radio" name="target" value="h">法人用
<ul id="ippan_area" class="active">
 <li>氏名<input type="text" name="name"></li>
 <li>住所<input type="text" name="address"></li>
 <li>電話番号<input type="text" name="tel"></li>
</ul>
<ul id="hojin_area" class="passive">
 <li>会社名<input type="text" name="company"></li>
 <li>部署名<input type="text" name="busho"></li>
 <li>担当者名<input type="text" name="tanto"></li>
 <li>住所<input type="text" name="address"></li>
 <li>電話番号<input type="text" name="tel"></li>
 <li>内線番号<input type="text" name="naisen"></li>
</ul>
```

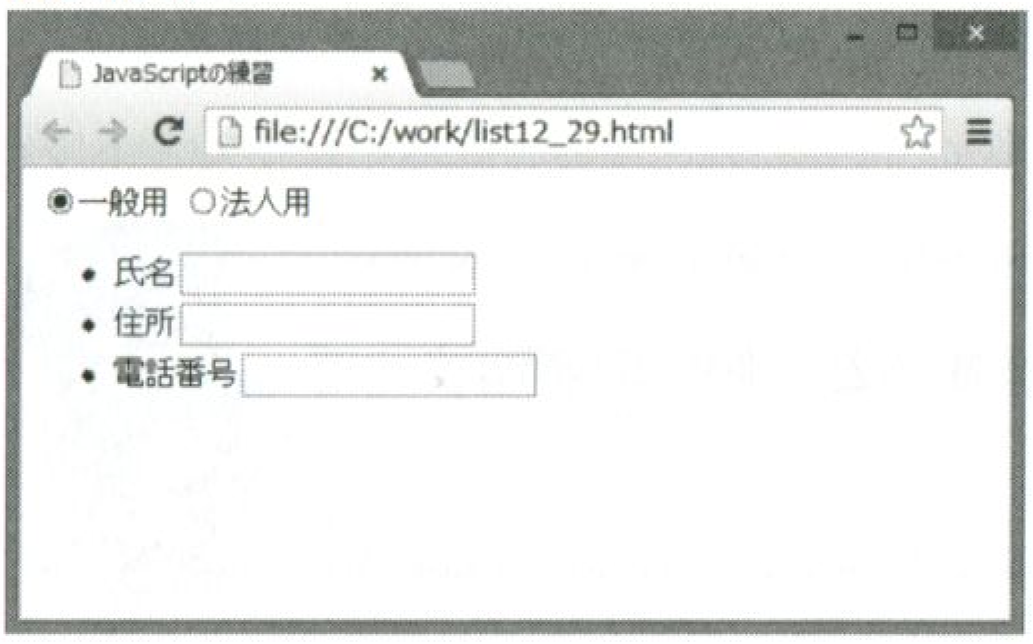
JavaScript

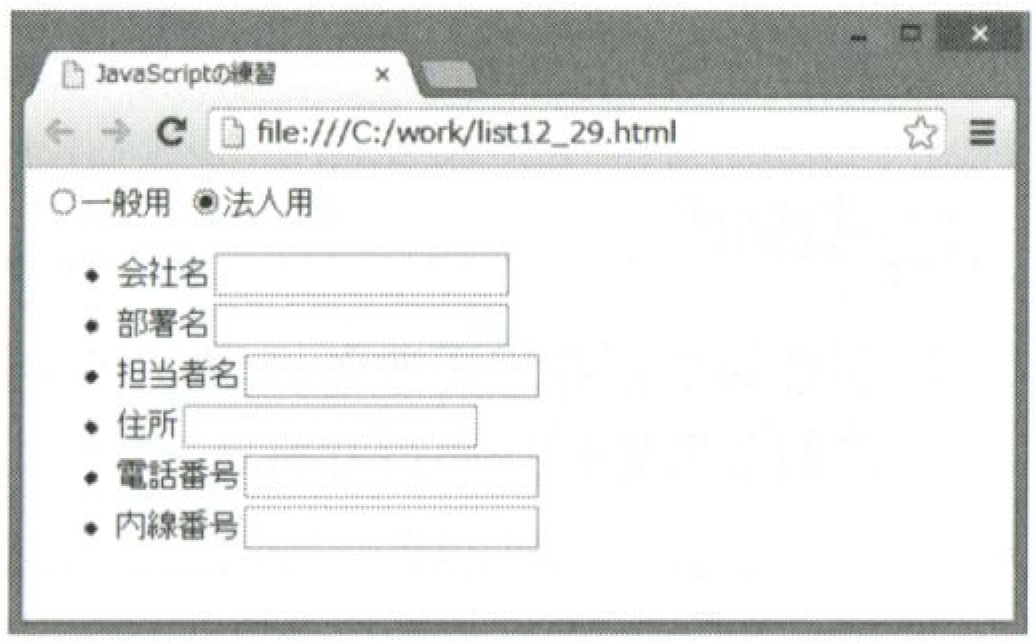
```
$(function() {
    $("input[name='target']").on("change", function(){
        $("#ippan_area, #hojin_area").toggleClass
("active passive");
    });
});
```

CSS

```
.active {
    display: block;
}
.passive {
    display: none;
}
```

■実行結果





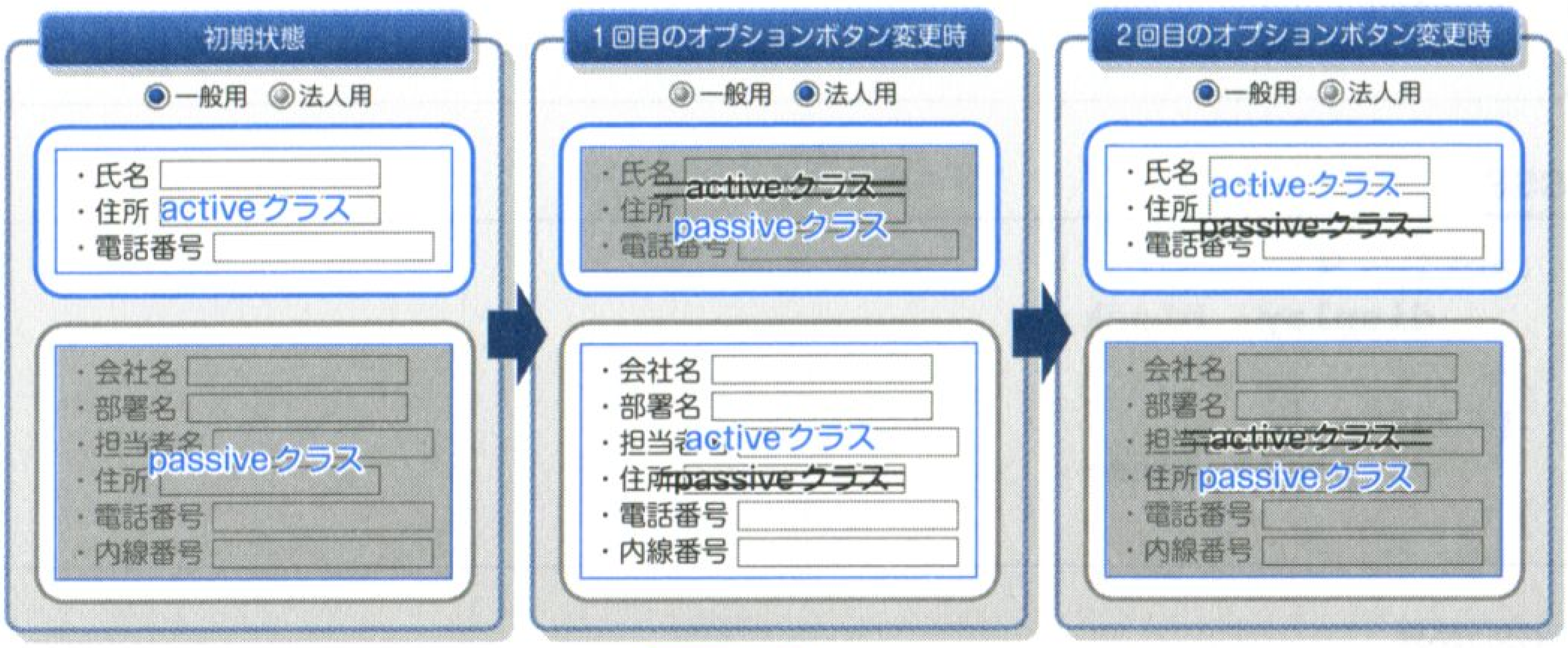
リスト12.29は、第10章で紹介した表示エリアを切り替えるスクリプト（233ページのリスト10.3）を、jQueryを利用して書き直したものです。まず、オプションボタンの選択状態が変更されたときの処理として匿名関数を登録しています。匿名関数の中では、2つの表示エリアに対して「active」クラスと「passive」クラスを交互に追加／削除しています。

```
$("input[name='target']").on("change", function(){
    $("#ippan_area, #hojin_area").toggleClass("active passive");
});
```

オプションボタンのチェック状態が変更されるたびに実行される関数

このページの初期状態では、一般用入力エリアには「active」クラスが、法人用入力エリアには「passive」クラスが設定されています。オプションボタンの選択状態が変更されると、一般用入力エリアから「active」クラスを削除し、

「passive」クラスを登録します。反対に法人用入力エリアからは「passive」クラスを削除して「active」クラスを登録します。要するにCSSクラスが現在登録されていれば削除し、削除されていれば登録する、という動きをしているのです。



実行イメージ

Point!

▶ 属性値などを変更するメソッドを実行すると、対象の要素すべてに対して処理が実行される。



■ Check Test 4

Q1 以下は、<button>要素がクリックされたタイミングで<p>要素の内容に「こんにちは」を設定するスクリプトです。空欄AとBを埋めて完成させてください。

```
$("button").[  A  ]("click", function(){ $("p").[  B  ] ⏎
("こんにちは"); });
```

Q2 以下は、<input>要素のname属性値に"hoge"を設定するスクリプトです。空欄AとBを埋めて完成させてください。

```
$("input").[  A  ]("[  B  ]", "hoge");
```

➲解答は巻末に掲載

12-5 要素の挿入

要素の追加／削除／置換

前節で学んだメソッドは、あらかじめ存在する要素の内容や属性を変更するものでした。ここではさらに高度なDOM操作として、HTML文書内に新たな要素を追加したり、既存の要素の削除をしたりする方法を学びます。要素ごとHTMLを書き換えることができるため、同じページを表示しているにもかかわらずまったく別のページが表示されたかのような視覚効果を与えることができます。

◆ appendメソッド、prependメソッド

要素内に新たな要素を追加します。引数には開始タグから終了タグまでのHTMLを記述します。appendメソッドは末尾、prependメソッドは先頭に要素を追加します。

文法 appendメソッド

```
要素.append("HTML要素");
```

文法 prependメソッド

```
要素.prepend("HTML要素");
```

リスト12.30 子要素の追加

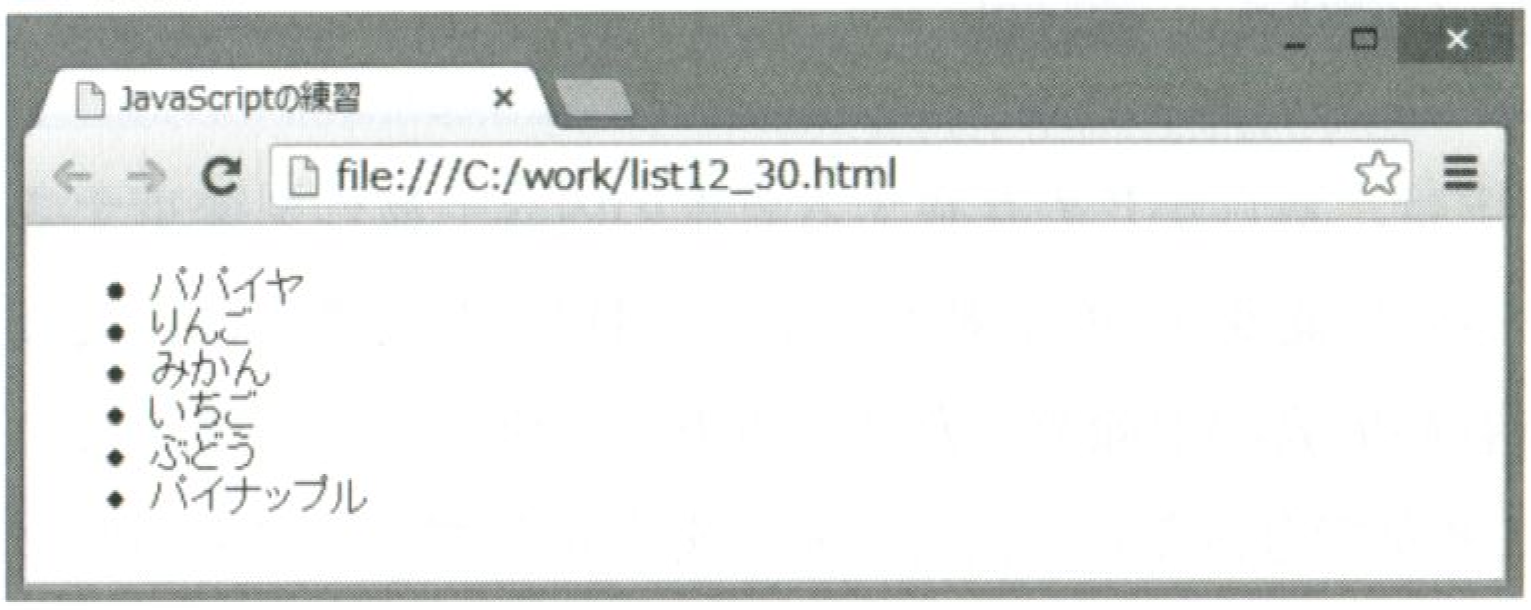
HTML (<body>要素内のみ抜粋)

```
<ul id="fruits">
 <li>りんご</li>
 <li>みかん</li>
 <li>いちご</li>
 <li>ぶどう</li>
</ul>
```

JavaScript

```
$(function() {
    $("#fruits")
        .append("<li>パイナップル</li>")
        .prepend("<li>パパイヤ</li>");
});
```

■実行結果



リスト12.30は対象の<ul>要素内に2つの要素を追加しています。スクリプトが実行されると、HTMLが次の状態に変更されます。

実行イメージ

```
<ul id="fruits">
  <li>パパイヤ</li>        ―― prependメソッドによって先頭に追加
  <li>りんご</li>
  <li>みかん</li>
  <li>いちご</li>
  <li>ぶどう</li>
  <li>パイナップル</li>    ―― appendメソッドによって末尾に追加
</ul>
```

◆ afterメソッド、beforeメソッド

要素の前後に新たな要素を追加します。appendメソッド、prependメソッドが子要素の追加だったのに対して、afterメソッド、beforeメソッドは兄弟要素を追加します。

文法 afterメソッド

```
要素.after("HTML要素");
```

文法 beforeメソッド

```
要素.before("HTML要素");
```

リスト12.31　兄弟要素の追加

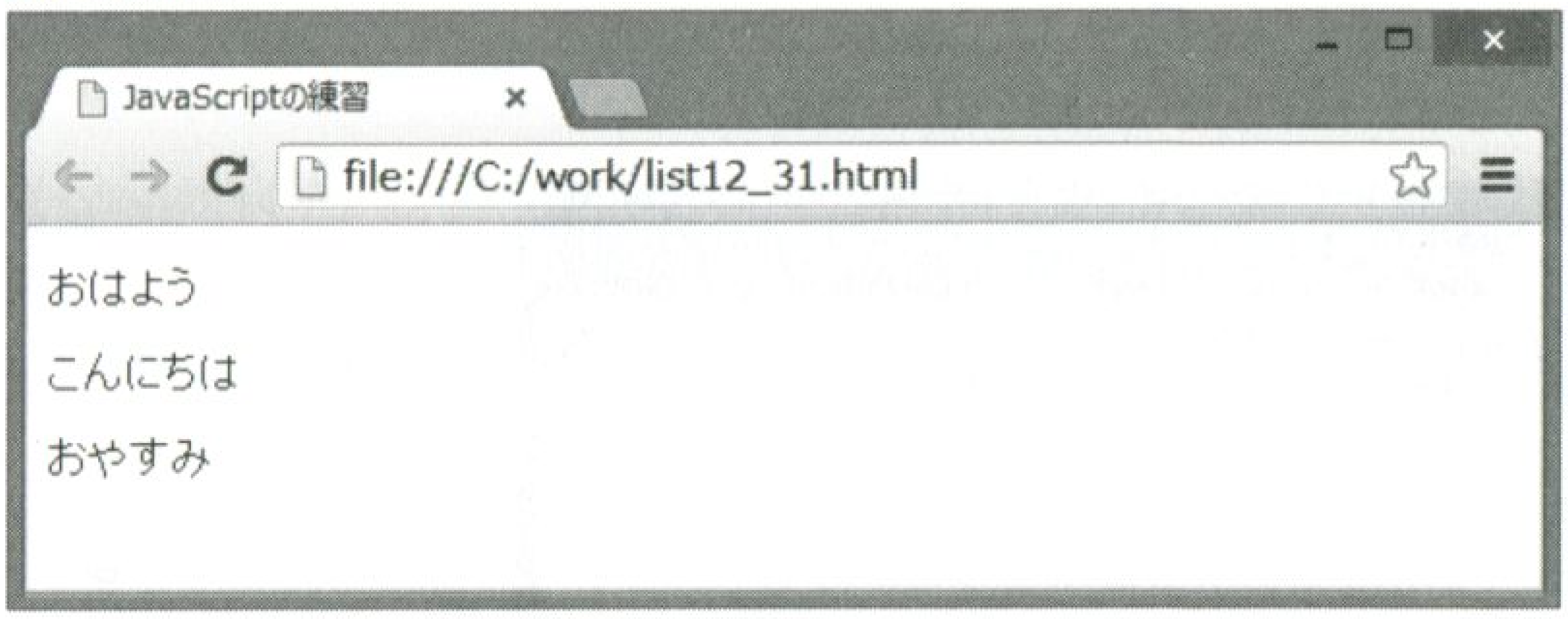
HTML　（<body>要素内のみ抜粋）

```
<p>こんにちは</p>
```

JavaScript

```
$(function() {
    $("p")
        .after("<p>おやすみ</p>")
        .before("<p>おはよう</p>");
});
```

■実行結果



リスト12.31は、対象の<p>要素の前後に2つの要素を追加しています。スクリプトが実行されると、HTMLが次の状態に変更されます。

実行イメージ

<p>おはよう</p> ―― beforeメソッドによって前に追加
<p>こんにちは</p>
<p>おやすみ</p> ―― afterメソッドによって後ろに追加

◇ emptyメソッド

要素が持つすべての子要素を削除します。実行後は、要素の内容は空になります。

文法　emptyメソッド

```
要素.empty();
```

リスト12.32　子要素の削除

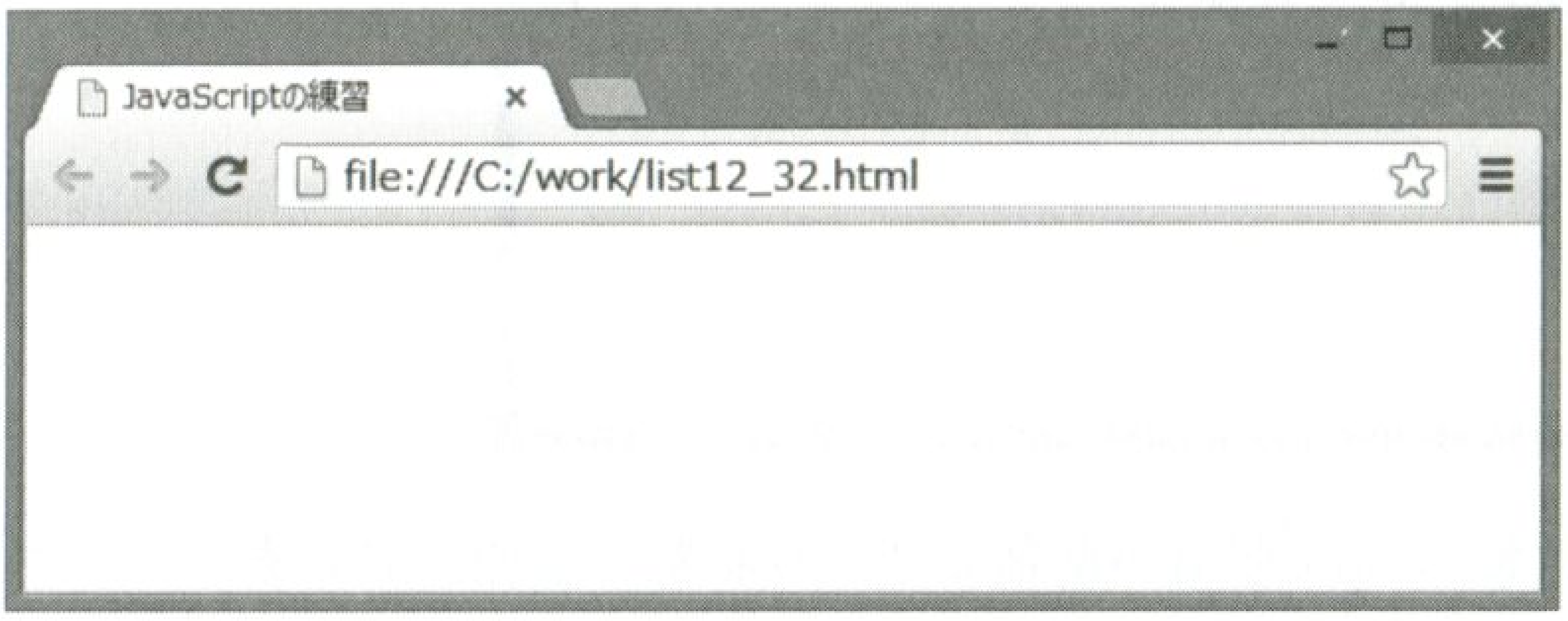
HTML　(<body>要素内のみ抜粋)

```
<ul id="fruits">
 <li>りんご</li>
 <li>みかん</li>
 <li>いちご</li>
 <li>ぶどう</li>
</ul>
```

JavaScript

```
$(function() {
    $("#fruits").empty();
});
```

■実行結果



リスト12.32は、対象の<ul>要素が持つ子要素をすべて削除しています。スクリプトが実行されると、HTMLが次の状態に変更されます。

実行イメージ

```
<ul id="fruits">
</ul>
```

emptyメソッドによって子要素がすべて消える

◆ replaceWithメソッド

要素を指定した別の要素で置き換えます。

文法 replaceWithメソッド

```
要素.replaceWith("HTML要素");
```

リスト12.33 要素の置換

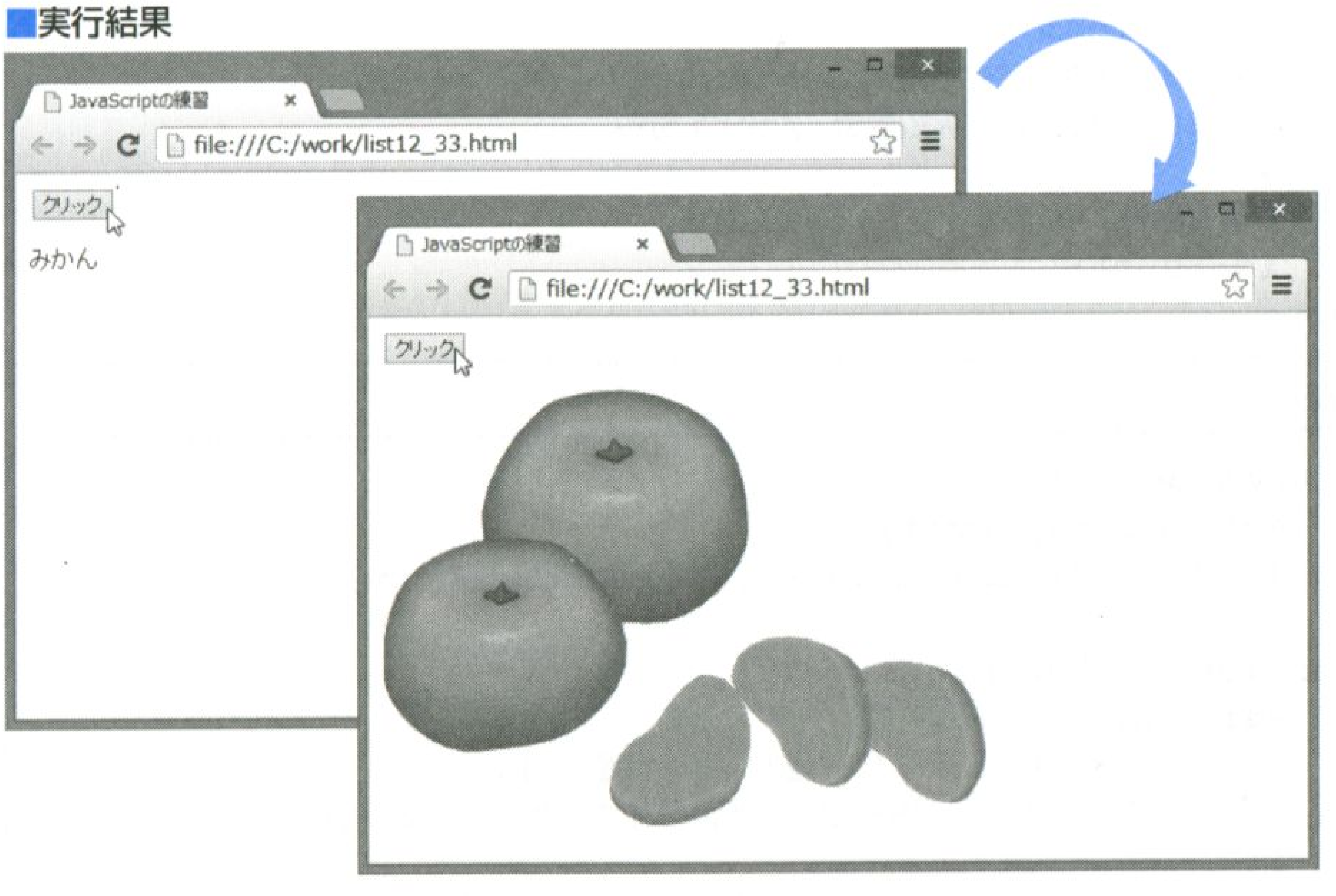
HTML （<body>要素内のみ抜粋）

```
<button id="btn">クリック</button>
<div><p>みかん</p></div>
```

JavaScript

```
$(function() {
  $("#btn").on("click", function() {
    $("p").replaceWith("<img src='images/orange.png'></img>");
  });
});
```

■実行結果



リスト12.33は、ボタンのclickイベントで<p>要素を<img>要素に置き換えています。ボタンがクリックされると、HTMLが次の状態に変更されます。

```
<div><img src='images/orange.png'></img></div>
```

<p>要素が<img>要素で上書きされる

DOM操作の応用

最後に、DOM操作の応用例としてセレクトボックスの操作方法を紹介します。リスト12.34は、メインカテゴリー一覧から項目を選択すると、サブカテゴリー一覧が一新されるスクリプトです。以降でスクリプトを行ごとに詳しく説明するため、JavaScriptのソースコードに行番号を入れてあります。

リスト12.34　セレクトボックスの操作

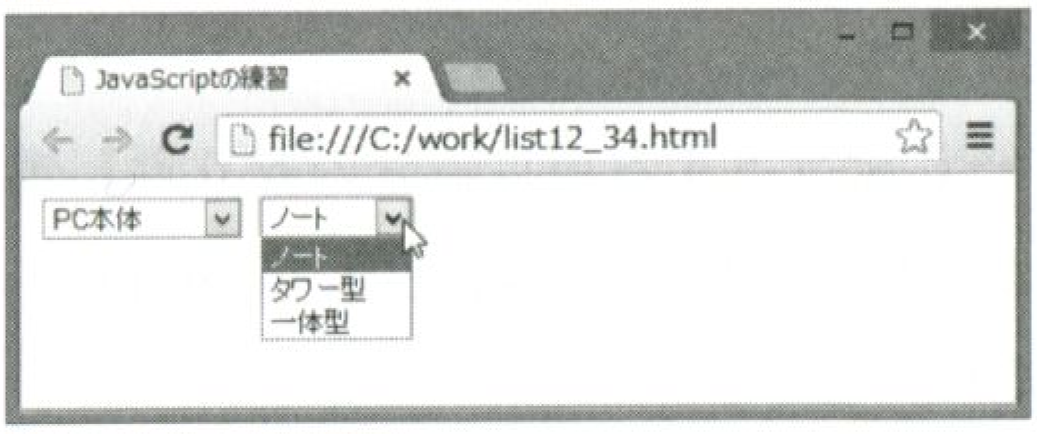
HTML　(<body>要素内のみ抜粋)

```
<!-- メインカテゴリー -->
<select id="main">
 <option value="0">PC本体</option>
 <option value="1">PC周辺機器</option>
</select>
<!-- サブカテゴリー -->
<select id="sub">
</select>
```

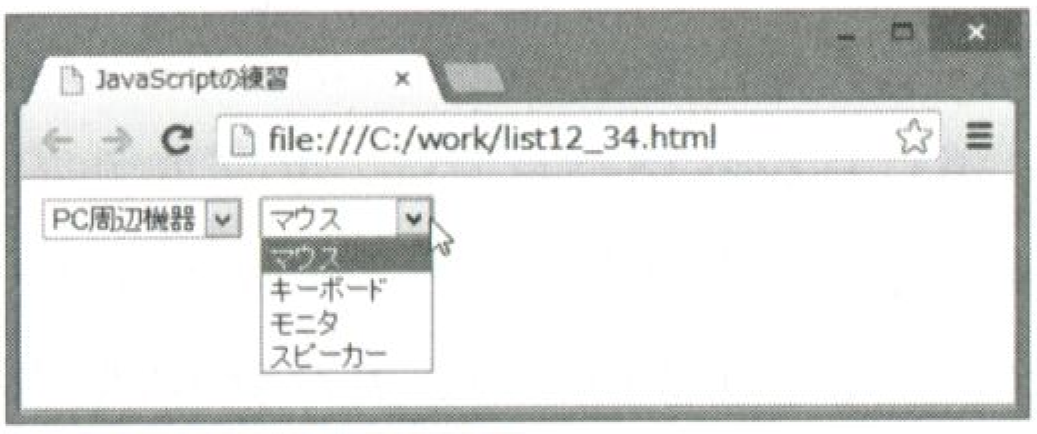
JavaScript

```
 1: $(function() {
 2:     setSubCategory();
 3:     $("#main").on("change", setSubCategory);
 4: });
 5: function setSubCategory() {
 6:     var subList = [
 7:         [ "ノート", "タワー型", "一体型" ],
 8:         [ "マウス", "キーボード", "モニタ", "スピーカー"]
 9:     ];
10:
11:     var i = $("#main > option:selected").val();
12:
13:     var sub = $("#sub");
14:     sub.empty();
15:
16:     for(var j=0; j < subList[i].length; j++) {
17:         sub.append("<option value='" + j + "'>" + ↵
    subList[i][j] + "</option>");
18:     }
19: }
```

■実行結果



「PC本体」選択時

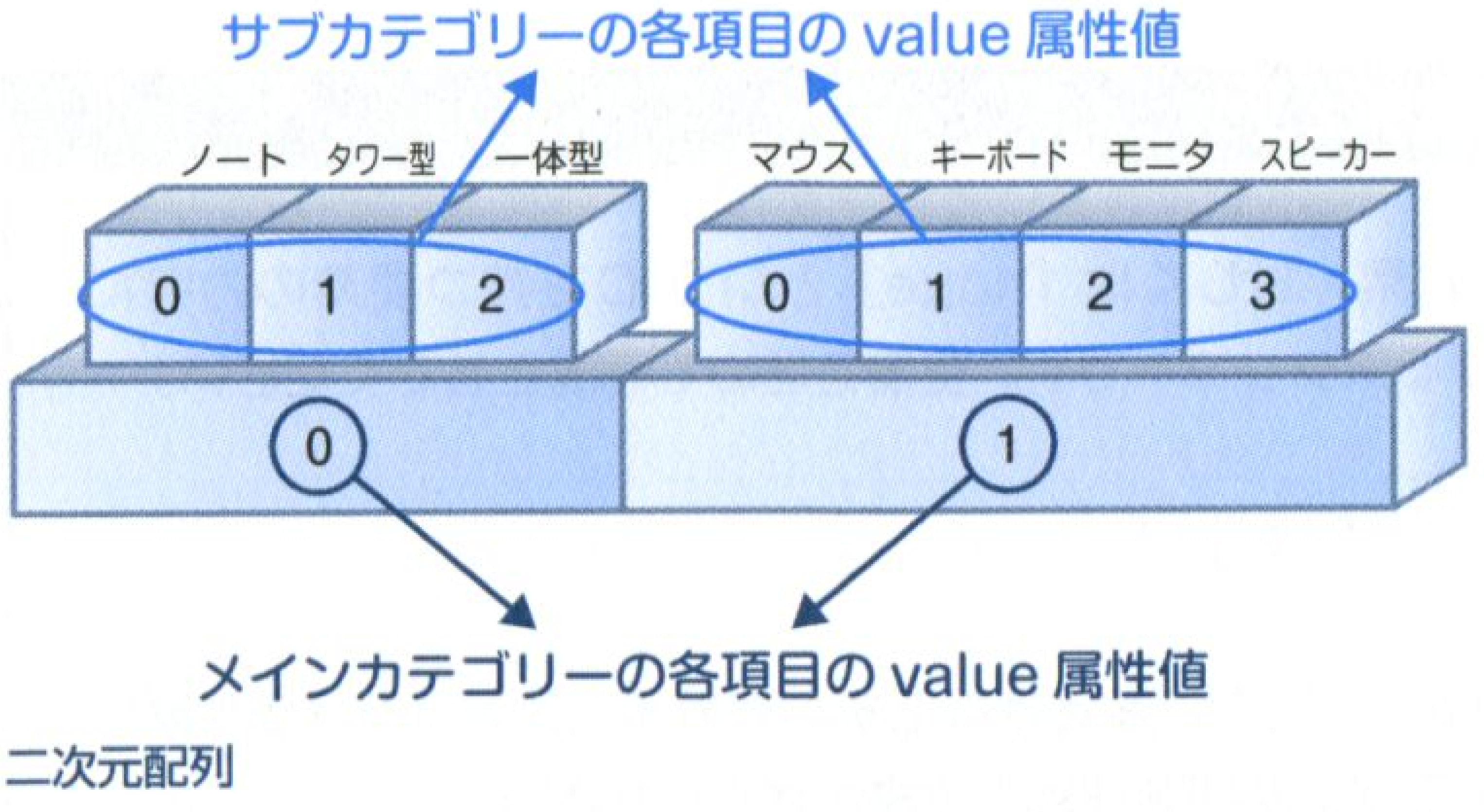


「PC周辺機器」選択時

まずはHTMLから見ていきましょう。メインカテゴリーとサブカテゴリーの2つのセレクトボックスがあります。メインカテゴリーには項目が設定されていますが、サブカテゴリーの項目は未設定です。

続いて、スクリプトの処理を追っていきます。1行目から4行目はWebページが表示されたときの処理です。setSubCategory関数を実行し、さらにメインカテゴリーのchangeイベントにsetSubCategory関数を登録しています。

5行目以降はsetSubCategory関数の処理です。6行目から9行目では、サブカテゴリーにセットする項目を二次元配列で保持しています。イメージとしては次のようになっています。



二次元配列

11行目で、メインカテゴリーの中から選択中の項目を取得し、value属性の値を変数iに代入しています。:selectedフィルタは選択されている項目のみを抽出します。「PC本体」選択されているときは0が、「PC周辺機器」が選択されているときは1が変数iに入ります。

13行目から14行目では、サブカテゴリーの項目をすべて削除しています。こ

の処理を入れておかないと、メインカテゴリーのchangeイベントが発生するたびにサブカテゴリーの項目がどんどん増えていってしまいます。

16行目から18行目では、サブカテゴリーにセットする項目数分の繰り返し処理を行なっています。メインカテゴリーが「PC本体」だった場合、appendメソッドで追加されるのは次の3つの要素です。

```
sub.append("<option value='" + 0 + "'>" + subList[0][0] + ⏎
"</option>");
sub.append("<option value='" + 1 + "'>" + subList[0][1] + ⏎
"</option>");
sub.append("<option value='" + 2 + "'>" + subList[0][2] + ⏎
"</option>");
```

```
sub.append("<option value='0'>ノート</option>");
sub.append("<option value='1'>タワー型</option>");
sub.append("<option value='2'>一体型</option>");
```

Point!

▶ 要素の追加、削除などの高度なDOM操作を行なうことができる。



この章の到達度チェック

Q1 以下は、<p>要素、もしくは「hoge」というIDを持つ要素の内容に"ABC"を設定するスクリプトです。空欄を埋めて完成させてください。

```
$("[　　　　]").html("ABC");
```

Q2 以下は、<button>要素に設定されたすべてのイベントハンドラを削除するスクリプトです。空欄を埋めて完成させてください。

```
$("button").[　　　　]();
```

Q3 以下は、<ul>要素内の<li>要素すべてにCSSクラス「list」を追加し、さらにインデックスが2番目の要素にはCSSクラス「highlight」も追加するスクリプトです。空欄を埋めて完成させてください。

```
$("ul > li").addClass("list")[      ] (":eq(2)"). ↵
addClass("highlight");
```

Q4 以下は、<button>要素をクリックするたびに、<div>要素にCSSクラス「box」を付けたり外したりするスクリプトです。空欄AとBを埋めて完成させてください。

```
$("button").on("[ A ]", function() {
    $("div").[ B ]("box");
});
```

Q5 以下は、<h1>要素を<h2>要素に置き換えるスクリプトです（ただし内容は変更しません）。空欄を埋めて完成させてください。

```
<h1>ABC<h1>
```

スクリプト

```
$("h1").[        ]("<h2>ABC</h2>");
```

Q6 以下は、リストの末尾に項目を追加するスクリプトです。空欄を埋めて完成させてください。

```
<ul>
 <li>パンジー</li>
 <li>ガーベラ</li>
 <li>チューリップ</li>
</ul>
```

スクリプト

```
$("ul").[        ]("<li>ひまわり</li>");
```

Q7 以下は、<ul>要素が持つ子要素をすべて削除し、<ul>要素の後ろに兄弟要素として<a>要素を追加するスクリプトです。空欄AとBを埋めて完成させてください。

```
$("ul").[ A ]().[ B ]("<a href='index.html'>リンク</a>");
```

➲解答は巻末に掲載

12

第13章

Ajax

この章で学ぶこと

- Ajaxとは
- 外部HTMLの取得
- クロスドメイン
- JSON、JSONP



● Ajaxは Webサイト間のデータ通信の手段 ●



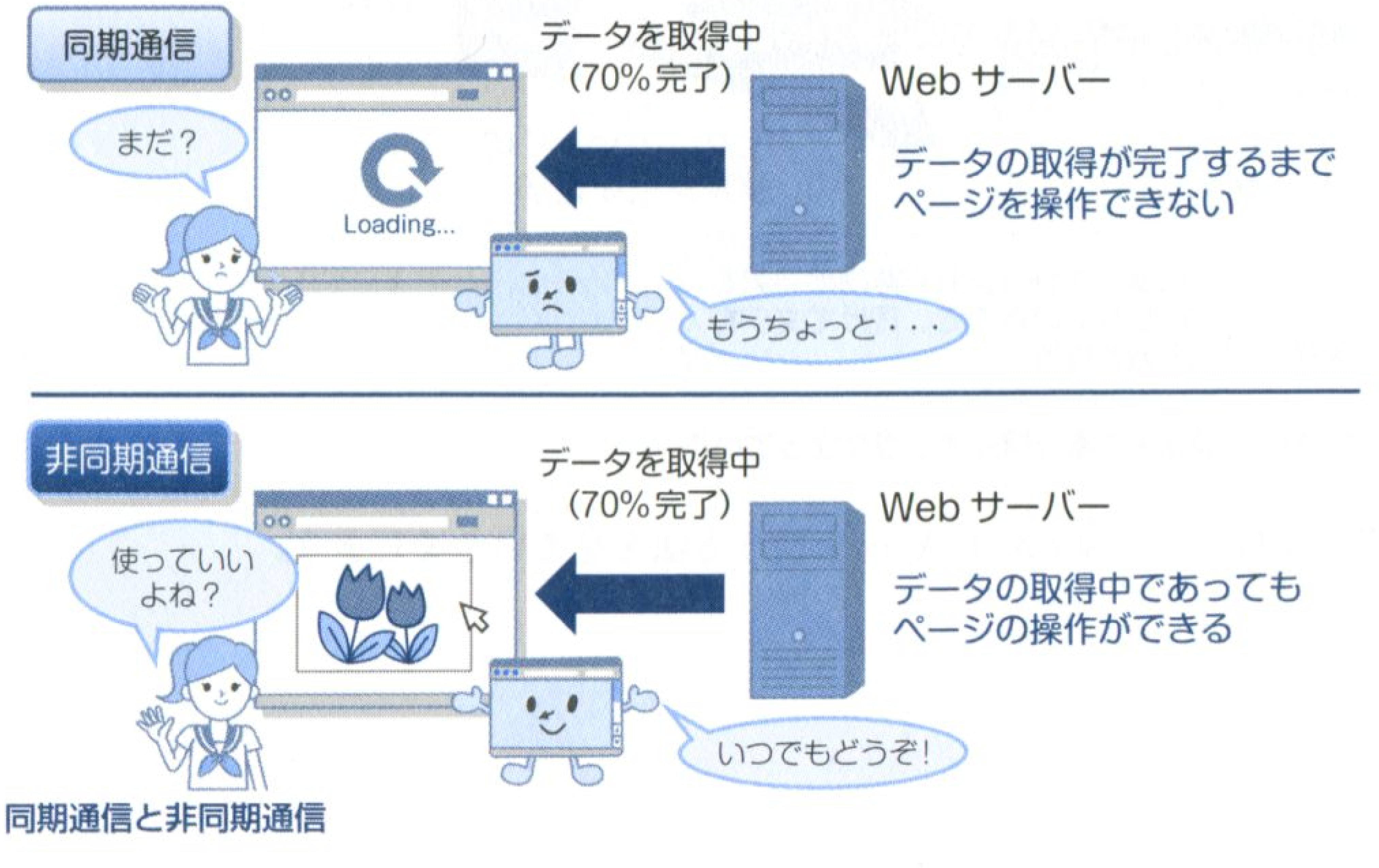
Ajaxは Webサイト間で情報を効率良くやり取りするための仕組みで、今日のWeb制作現場では欠かせません。JavaScriptでAjaxの処理を記述すると複雑になりがちですが、jQueryを使えば簡潔に記述できます。

13 1 Ajaxの基礎

Ajaxとは

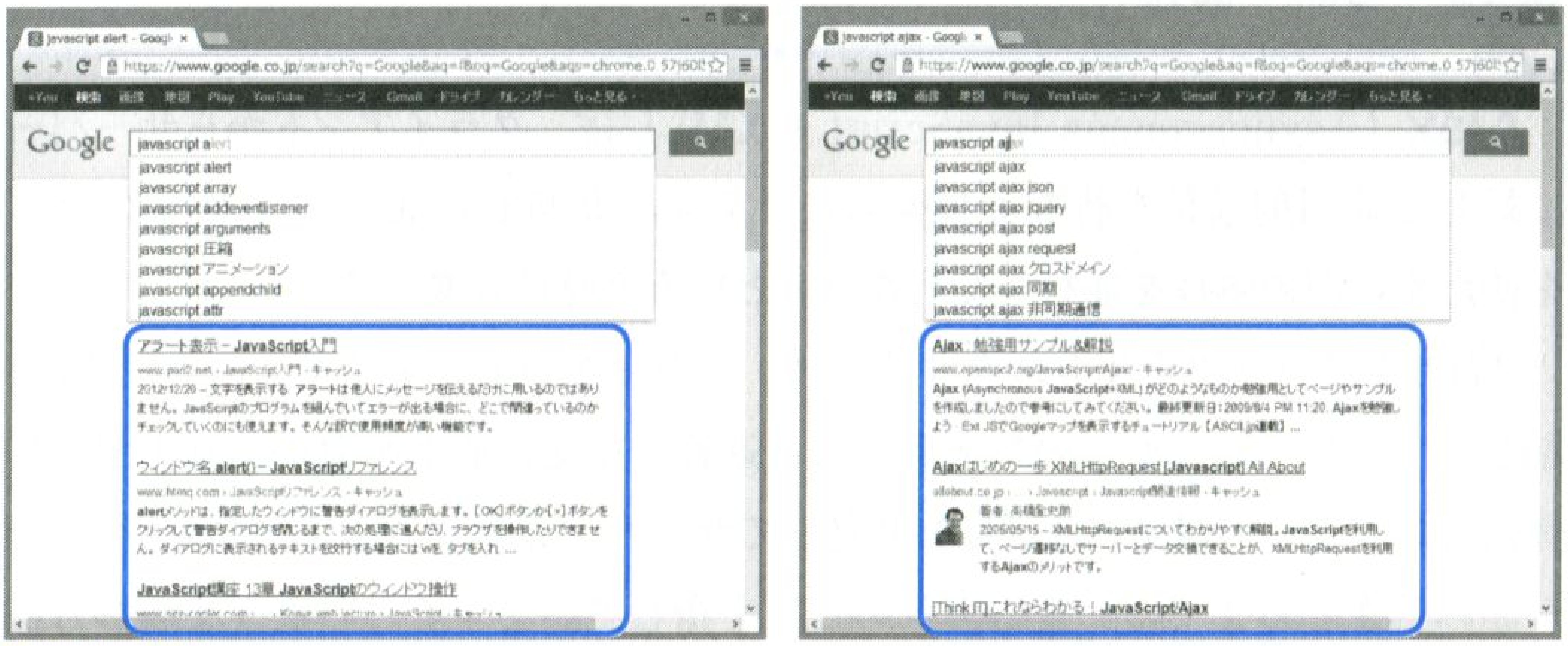
Ajax（エイジャックス）（Asynchronous JavaScript + XML）は、クライアントからサーバーに対して非同期通信を行なう技術のことです。非同期とは、「ある処理の完了を待たずに別の処理を並行して行なう」という意味の言葉です。

Web上でデータをやり取りするには、クライアントからリクエスト（要求）を送って、サーバーからのレスポンス（返答）を受け取る必要があります。具体的には、ブラウザからURLを指定してサーバーにアクセスし、Webページが読み込まれるまでの一連の流れのことを指しています。このような通信方法を同期通信と呼びます。同期通信では、クライアントはサーバーからのレスポンスが返ってくるのをひたすら待ち続けるしかありません。しかし、非同期通信を利用すれば、レスポンスを待機し続ける必要はありません。その間ユーザーはブラウザに表示されたWebページを自由に操作できます。しかも現在のWebページを見ているままで移動することもなく、別のWebページから情報を取得できるのです。



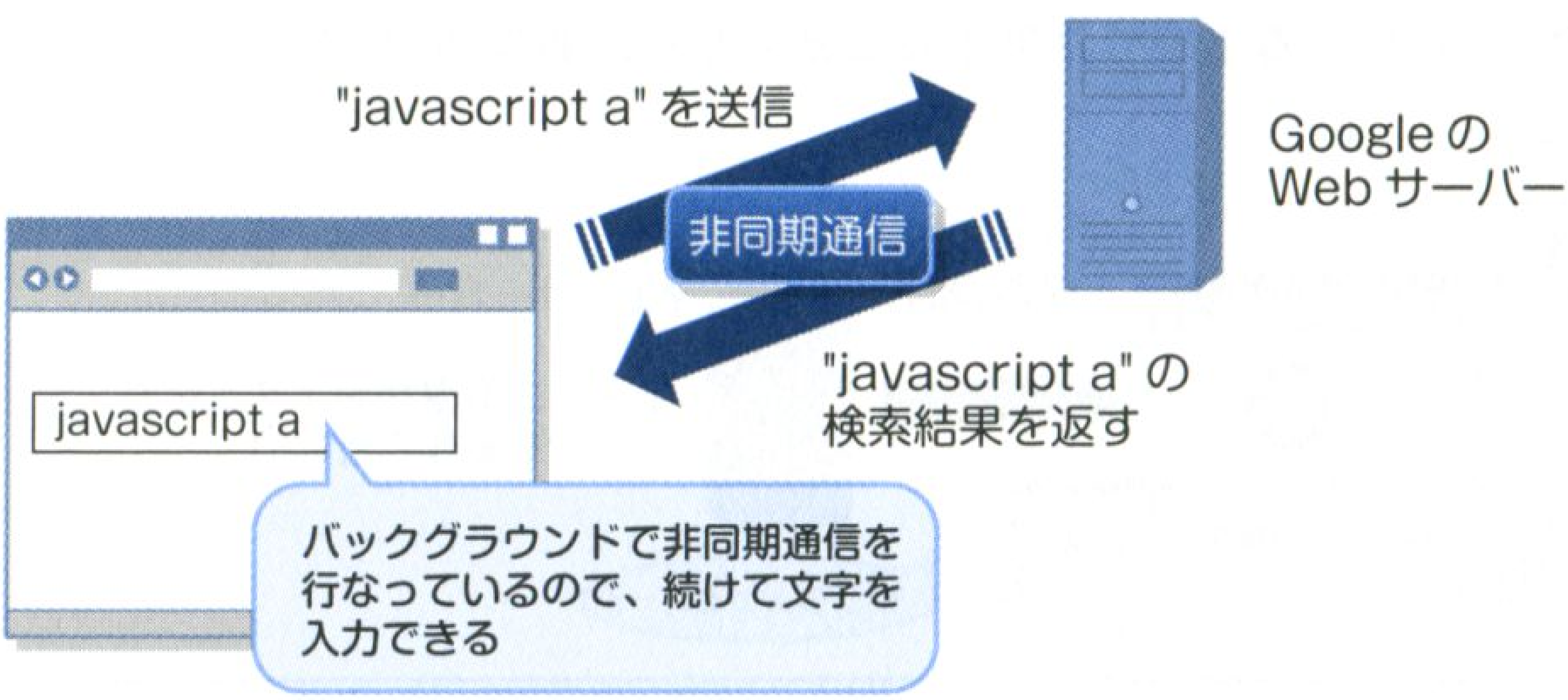
同期通信と非同期通信

Ajaxは、さまざまなWebサイトで利用されています。代表的なのものがGoogle検索（http://www.google.co.jp/）です。たとえば、検索キーワードに「javascript a」まで入力すると、検索ボタンを押さなくても検索結果が表示されます。さらにその後ろに続けて「j」を入力すると、先ほどとは別の検索結果が表示されます。



Google検索

このような、ユーザーが入力をしている裏側でデータを取得してきてくれる快適なシステムは、Ajaxの恩恵によるものなのです。



Google検索はAjaxの非同期通信で成り立っている

以降では、JavaScriptでAjaxを使う方法を見ていきましょう。

Ajaxによる HTMLの読み込み

あるページの内容を他のページにも表示したいときは、AjaxでHTMLを読み込むloadメソッドが便利です。loadメソッドは、読み込んだHTMLを要素に挿入します。

文法 loadメソッドの書式

```
要素.load("別ページのURL");
```

リスト13.1は、info.htmlからHTMLを取得し、main.htmlの<div>タグ内に挿入するスクリプトです。

リスト13.1 指定したページの内容を読み込む

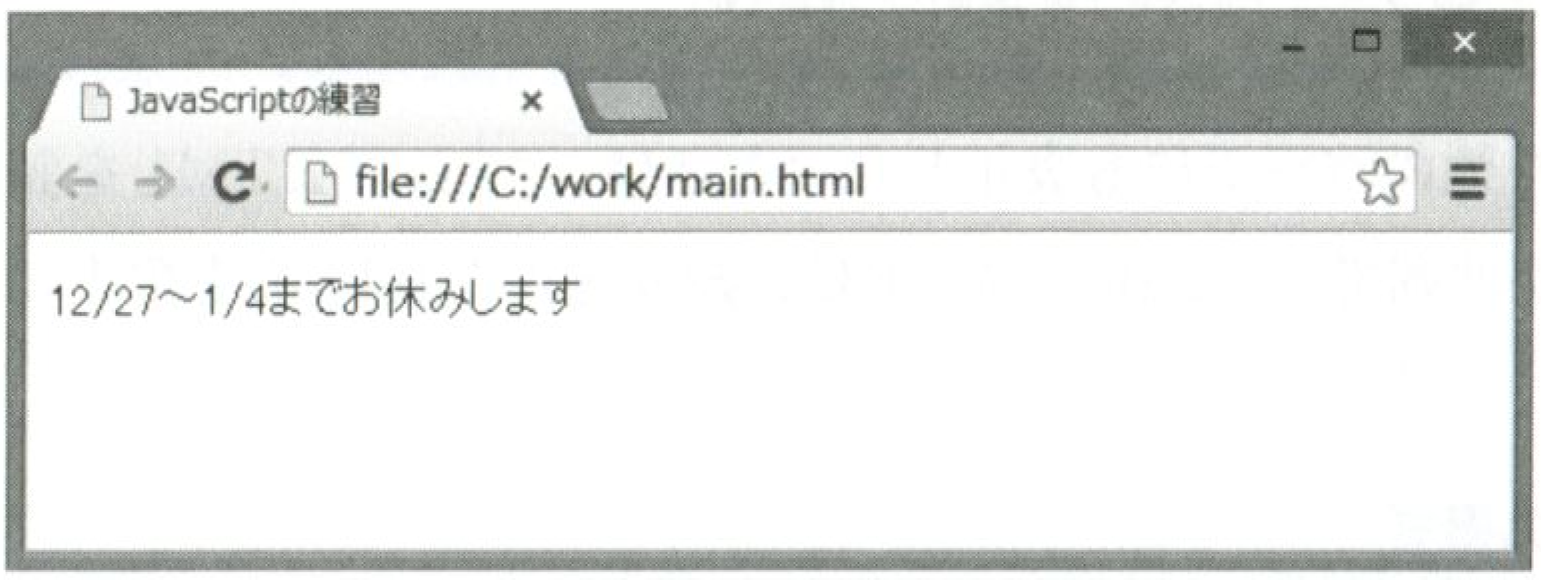
HTML1 main.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/ ⏎
jquery/1.9.1/jquery.min.js"></script>
  <script>
   $(function() {
     $("#infoArea").load("info.html");
   });
  </script>
 </head>
 <body>
  <div id="infoArea"></div>
 </body>
</html>
```

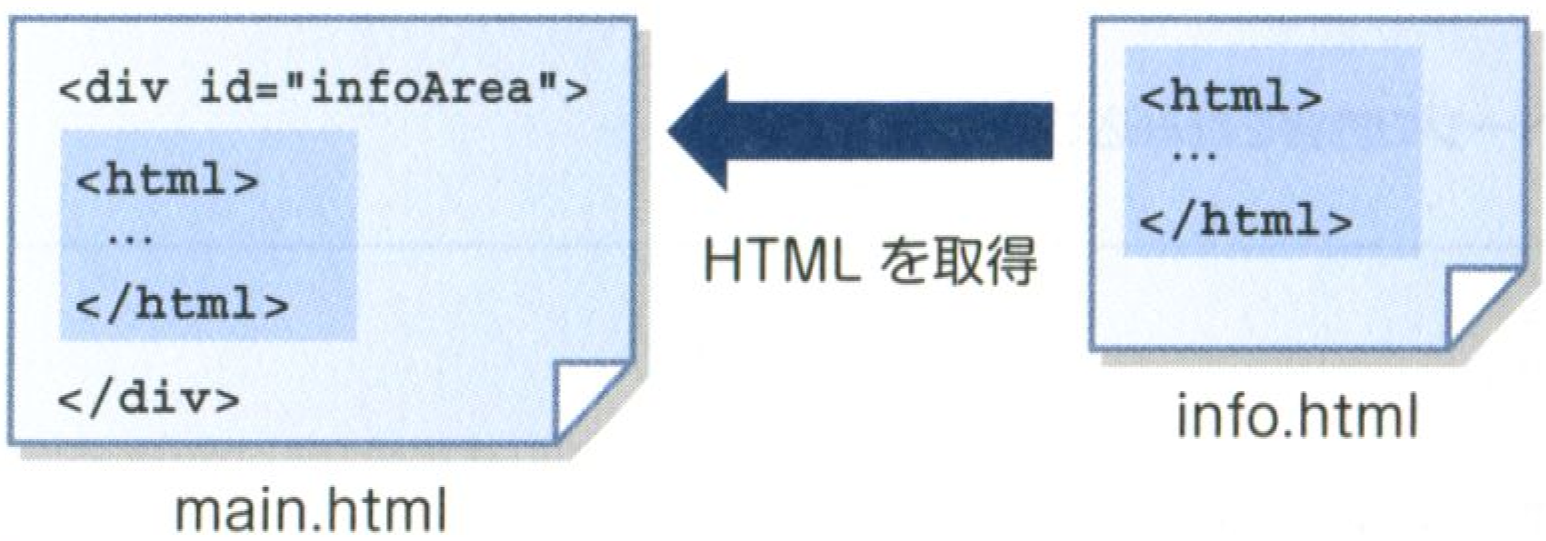
HTML2 info.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
 </head>
 <body>
  <p>12/27～1/4までお休みします</p>
 </body>
</html>
```

■実行結果



main.htmlを表示すると、main.htmlはinfo.htmlからHTMLを非同期で取得してページ内に表示します。



実行イメージ

Note

リスト13.1では、info.htmlの<title>や<meta>などの不要な要素も取得してしまっています。loadメソッドはURLの後ろにスペースを空けてセレクタを指定することもできるため、<p>要素のみを取得したいのであれば次のように記述します。

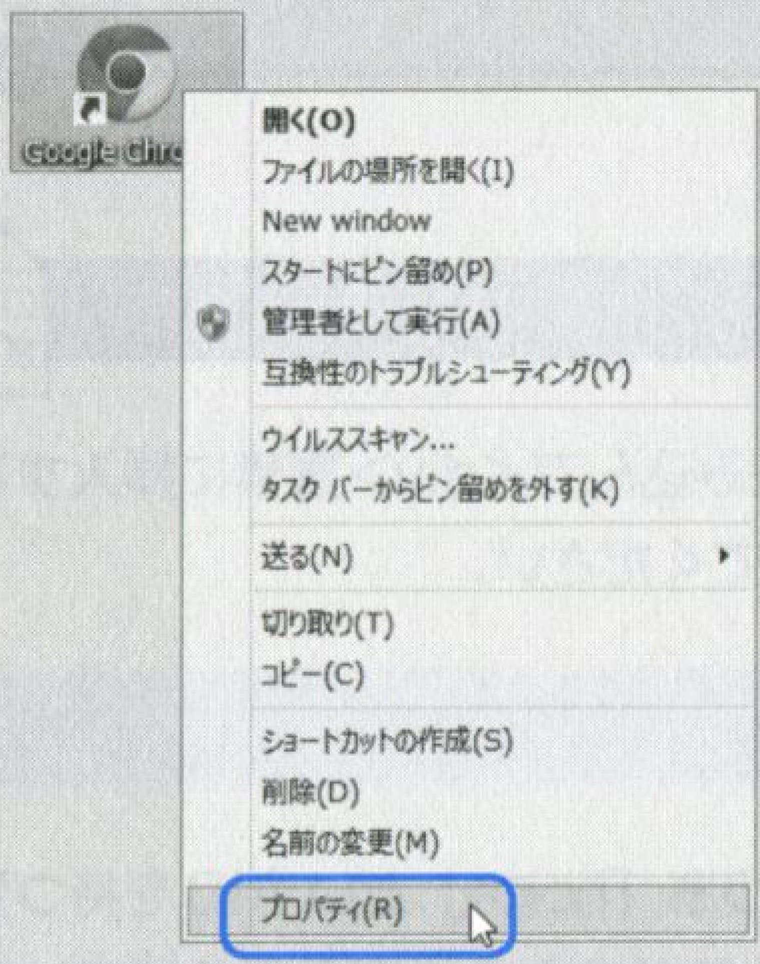
```
$("#infoArea").load("info.html p");
```

注意

loadメソッドでエラーが発生した場合

本書をお読みのみなさんは、ローカルのパソコン上でJavaScriptの実行確認をされている方が多いでしょう（ローカルとは、サーバーではなく自分のパソコンという意味です）。IEやFirefoxでは問題ありませんが、Chromeを使っている場合、loadメソッドでローカルファイル（自分のパソコンのファイル）を取得しようとするとエラーが発生します。その場合は、次のようにして対処してください。

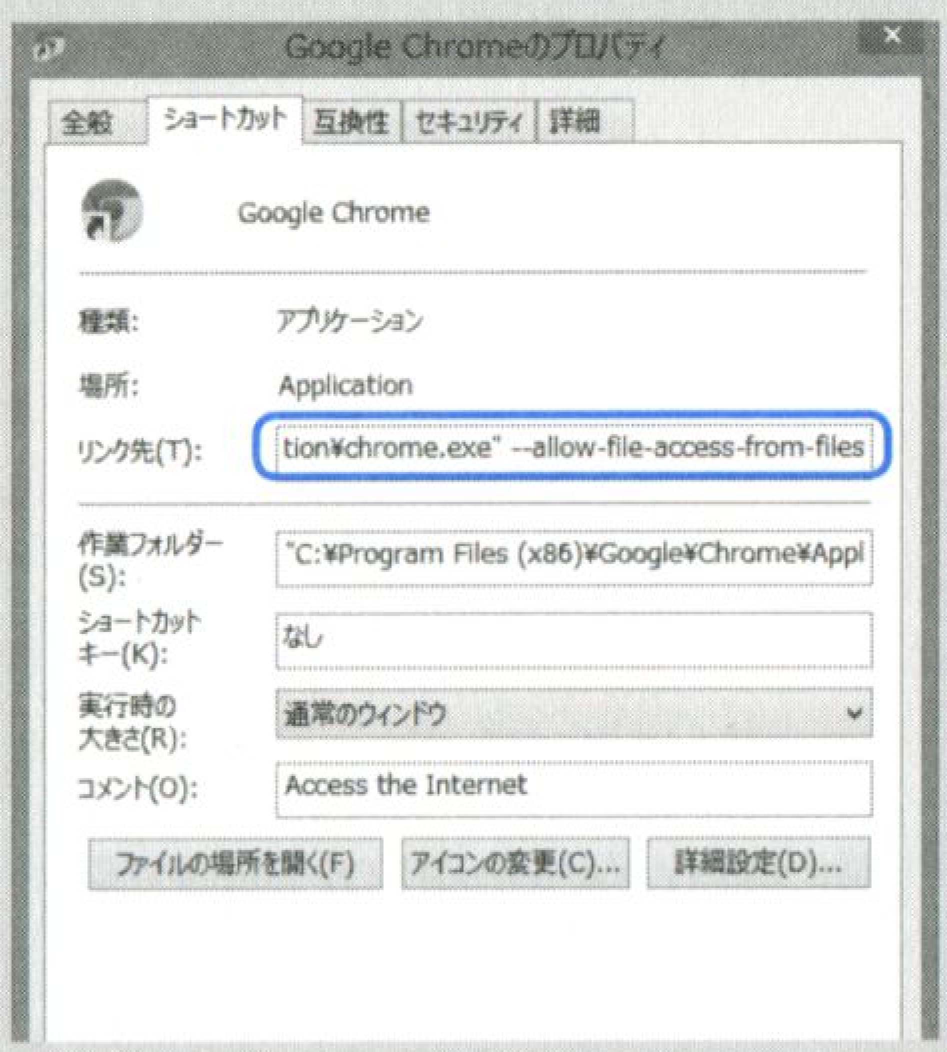
❶ Chromeのショートカットアイコン の上で右クリックして［プロパティ(R)］を選択します。



❷ Chromeのプロパティ画面が開くので、「リンク先」のchrome.exeの後ろに以下を指定します（先頭に半角スペースが必要です）。

```
chrome.exe" --allow-file-access-from-files
```

exe"と-の間に半角スペースが必要



❸ Chromeを開いている場合はいったん閉じて、ショートカットアイコンから再起動します。

もちろん、Web上にHTMLファイルをアップロードすれば、どのブラウザからでも問題なく実行することができます。

Point!

▶ AjaxはJavaScriptで非同期通信を実現する技術である。



■ Check Test 1

Q1 以下は、sample.htmlのHTMLを読み込んで<div>要素に挿入するプログラムです。空欄を埋めて完成させてください。

```
$("div").[      ]("sample.html");
```

Q2 sample.htmlから取得したHTMLのうち「hoge」というIDを持つ要素のみ<div>要素に挿入するには、Q1をどのように変更すれば良いですか？

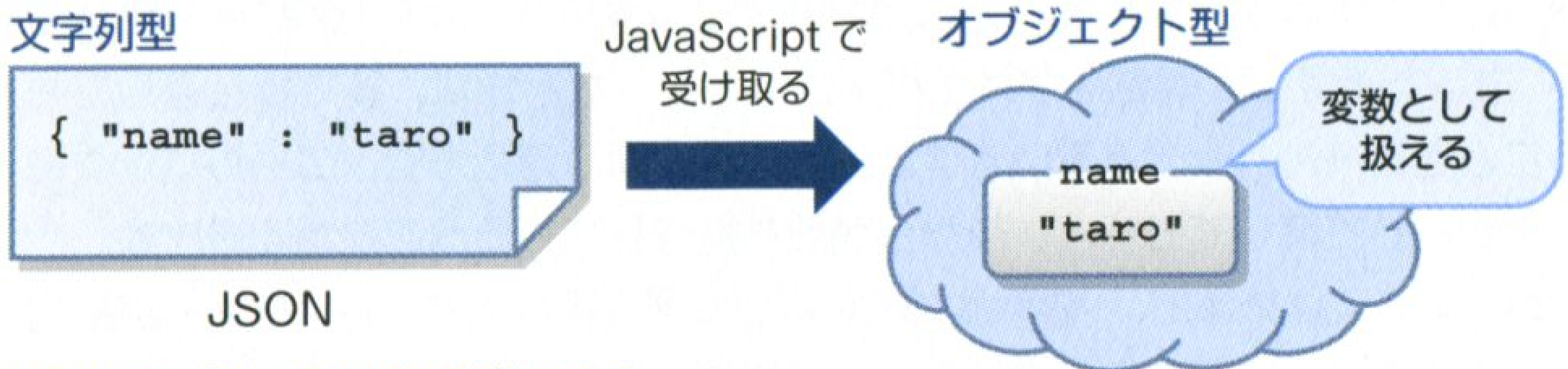
➲解答は巻末に掲載

13 2 JSONとJSONP

JSONとは

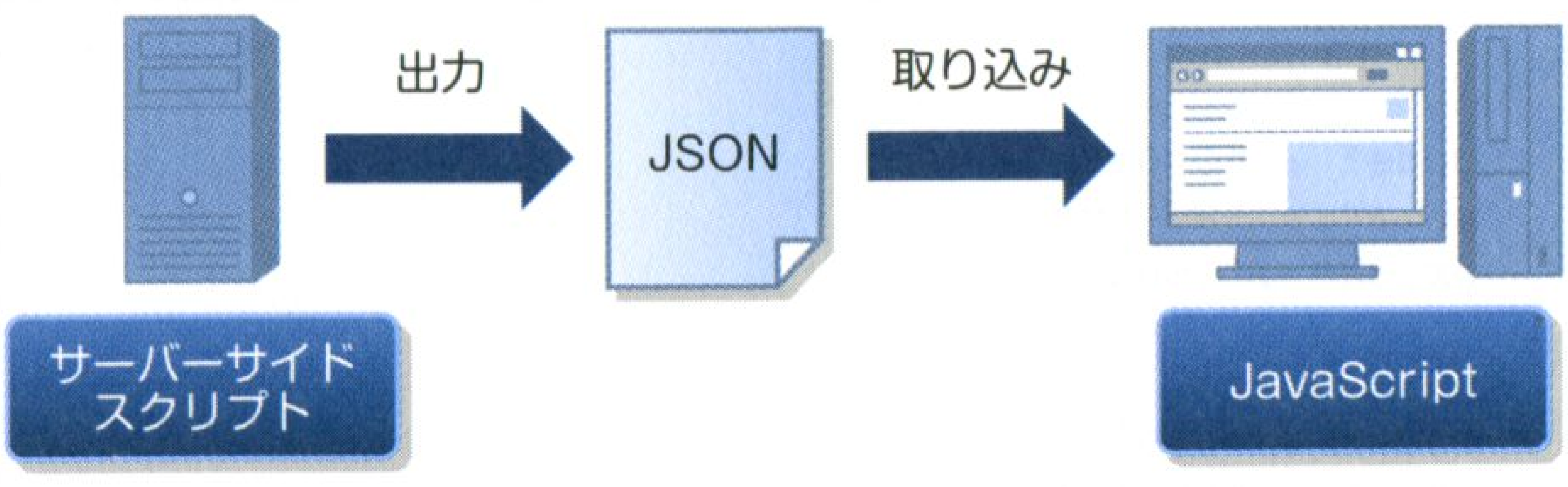
Web上で利用されるデータの記述形式と言えばXMLが有名ですが、それよりももっと扱いやすいJavaScriptのための記述形式があります。

JSON(ジェイソン)(JavaScript Object Notation)は、データのフォーマット(記述形式)の一種です。JavaScriptでデータ通信を行なう際のフォーマットとして開発されました。JSON自体は単純な文字列ですが、JavaScriptのオブジェクト型に変換して利用することができます。



JSONはオブジェクト型に変換される

JSONは書き方が簡単なので、JavaScriptはもちろん、他のスクリプト言語でも楽に扱うことができます。サーバーサイドスクリプトからJavaScriptへデータを渡す場合には非常に重宝します。本章では、サーバーサイドスクリプトで作られたWeb APIという機能を使ってサーバーからデータを取得するJavaScriptを作成します(Web APIの詳細は後述します)。



JSONを用いたサーバースクリプトとの連携

JSONの書き方と取得方法

JSONの記述方法は、JavaScriptの配列またはオブジェクトとほぼ同じですが、必ずプロパティ名をダブルクォート（"）で囲む必要があります。シングルクォート（'）は使えないので注意してください。JSONデータは、getJSONメソッドで取得できます。

文法 getJSONメソッド

```
$.getJSON("JSONデータのURL", コールバック関数);
```

第2引数の**コールバック関数**には、JSONの取得が完了したときに実行する関数を指定します。この関数には、引数としてJSONデータが渡されます。

> **Note** ある処理の完了後に実行される関数のことをコールバック関数と呼びます。たとえば、電話をかけた相手が不在だったときに、後で折り返してもらうことをコールバックと言いますよね。それと同じ考え方で、コールバック関数を引数として渡しておくと、後から関数を実行してもらうことができます。今回の処理に当てはめると、「❶getJSONメソッドを呼び出す」→「❷getJSONメソッドの処理が完了するのを待つ」→「❸コールバック関数が実行される」という流れです。

では、JavaScriptでJSONを取得する例を見てみましょう。

リスト13.2 JSONファイルの取得

JSON item.json

```
[
    {
        "code" : "0001",
        "name" : "EV-02",
        "price" : 1500
    },
    {
        "code" : "0002",
        "name" : "NR-07",
```

```
            "price" : 2500
        },
        {
            "code" : "0003",
            "name" : "WR-03",
            "price" : 3500
        }
]
```

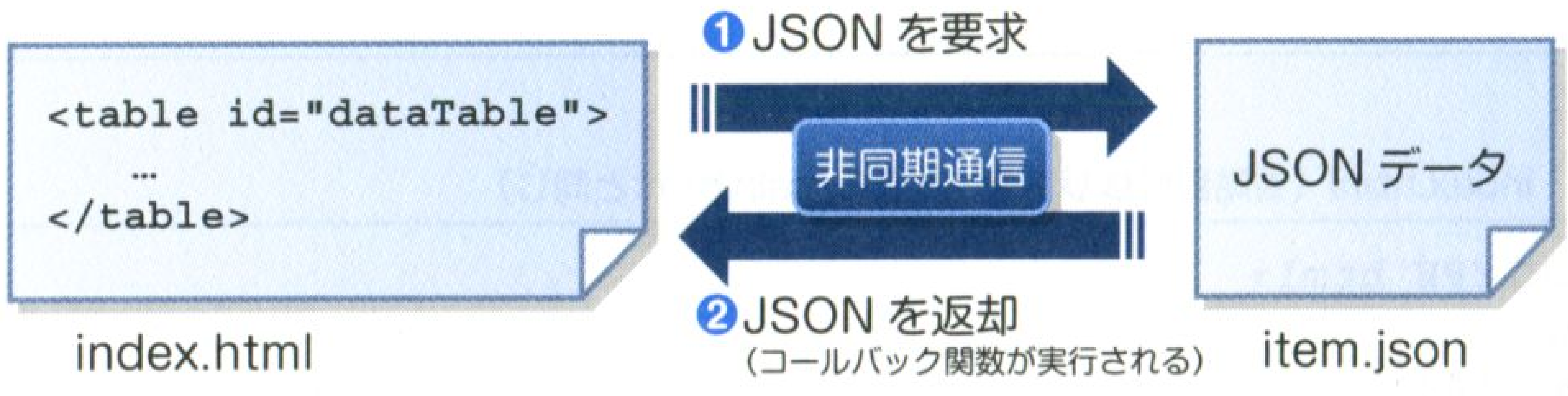
HTML index.html（省略部分はリスト13.1のmain.htmlと同じ）

```
<!DOCTYPE html>
  (略)
  <script>
   $(function() {
     $.getJSON("item.json",
       function(data) {
         for(var i=0; i<data.length; i++) {
           $("#dataTable").append("<tr></tr>")
             .find("tr:last")
             .append("<td>" + data[i].code + "</td>")
             .append("<td>" + data[i].name + "</td>")
             .append("<td>" + data[i].price + "</td>");
         }
       }
     );
   });
  </script>
 </head>
 <body>
  <table id="dataTable">
   <tr><th>code</th><th>name</th><th>price</th></tr>
  </table>
 </body>
</html>
```

■実行結果

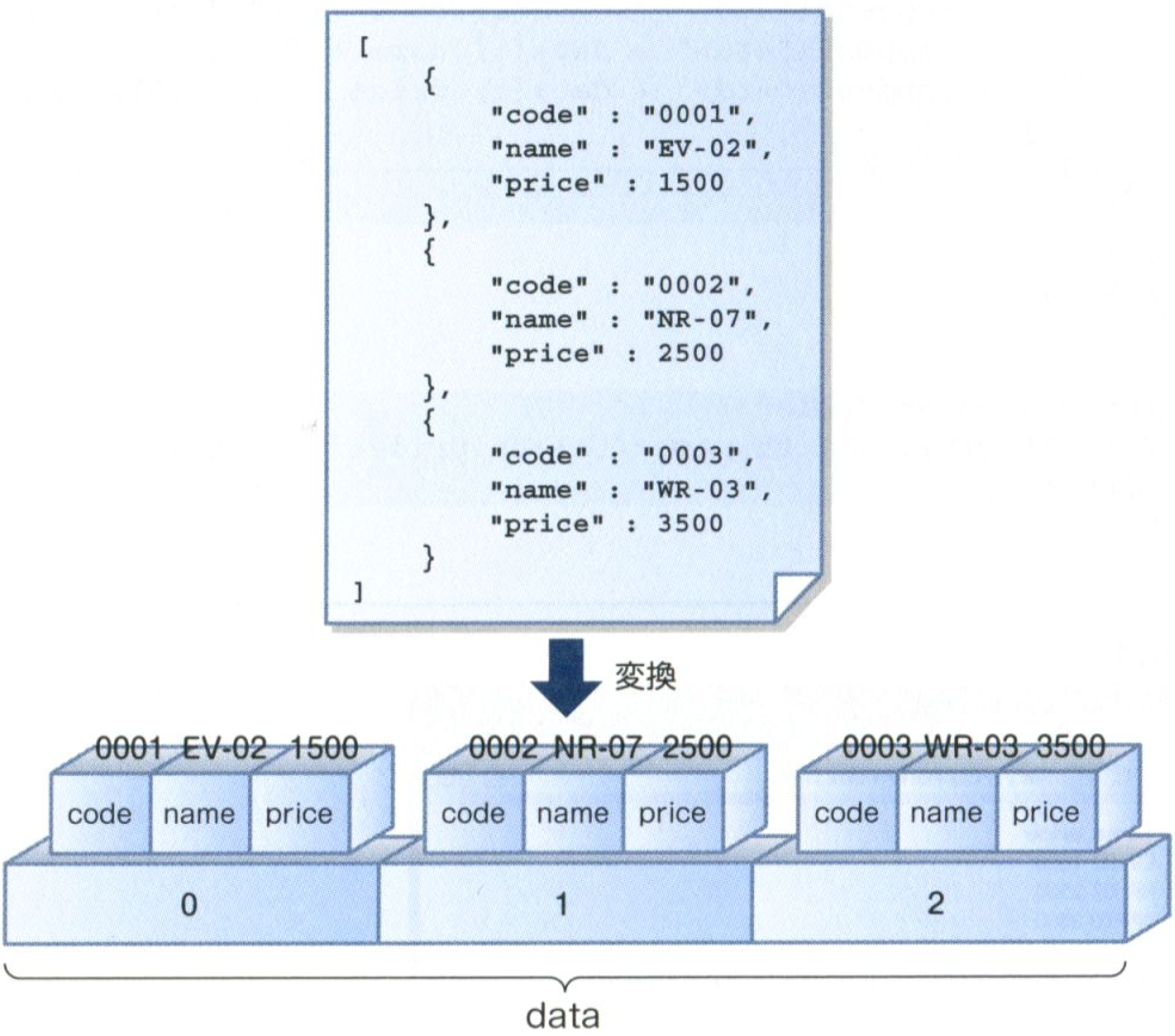


リスト13.2では、3件のオブジェクトを要素として持つ配列形式のJSONを取得しています。スクリプトの❶の部分がコールバック関数です。JSONが取得されるとJavaScriptのデータに変換され、コールバック関数の引数dataに渡されます。関数の中でテーブルのデータ行を生成し、受け取ったデータをセットしています。



リスト13.2の処理の流れ

このスクリプトでは、次のようにJSONデータからJavaScriptデータへの変換が行なわれています。



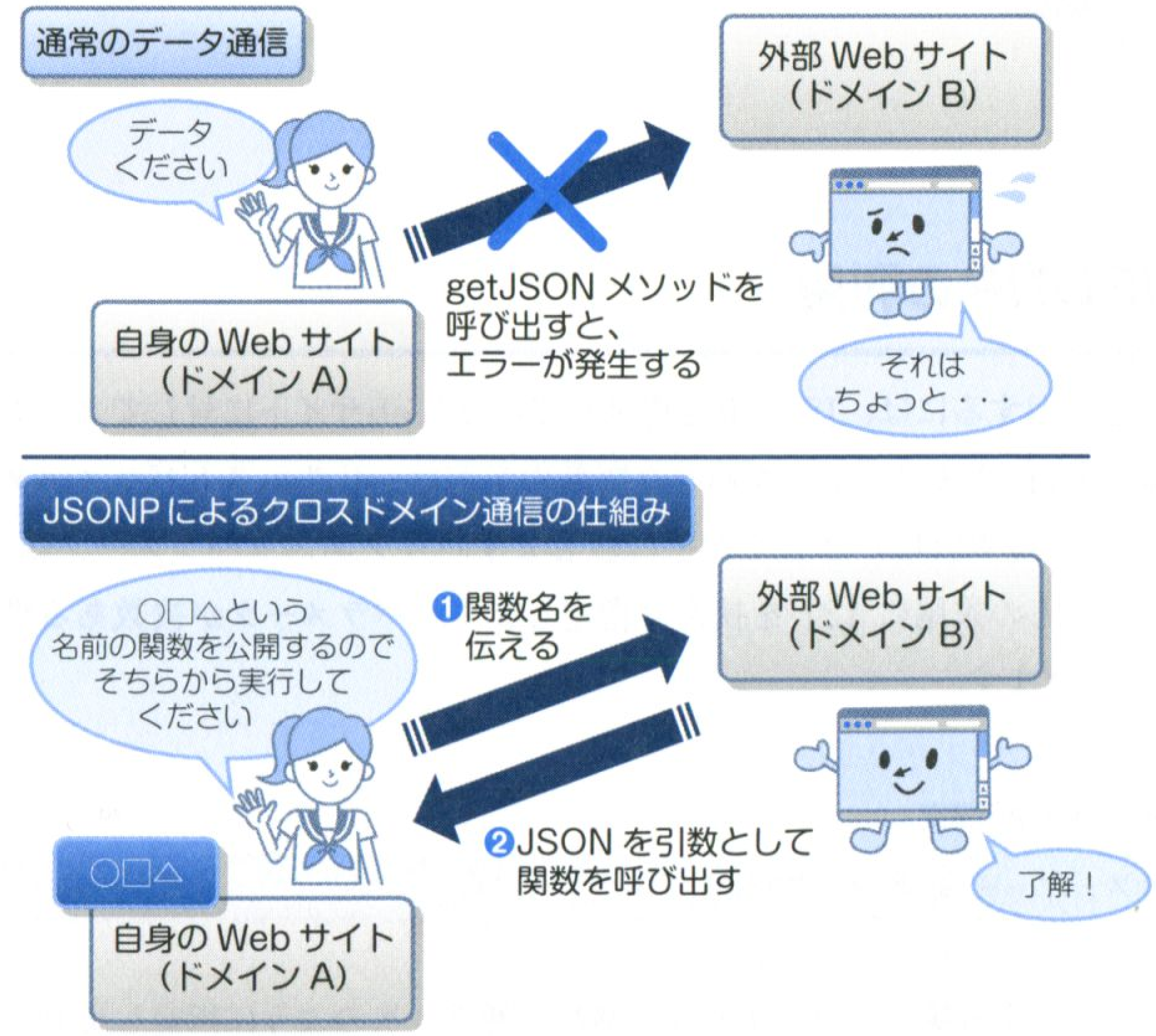
リストJSONデータからJavaScriptデータへの変換

JSONPとは

JavaScriptでは、セキュリティ上の問題で異なるドメインのサイトからデータを取得できない仕様になっています。しかし、JSONP（ジェイソンピー）という方法でデータをやり取りすれば、別ドメインのWebサイトからでもデータを受け取ることができるようになります。

用語解説
【ドメイン】
サーバーに付けられたインターネット上の名前をドメインと呼びます。また、異なるドメイン間でデータ通信を行なうことをクロスドメインと呼びます。

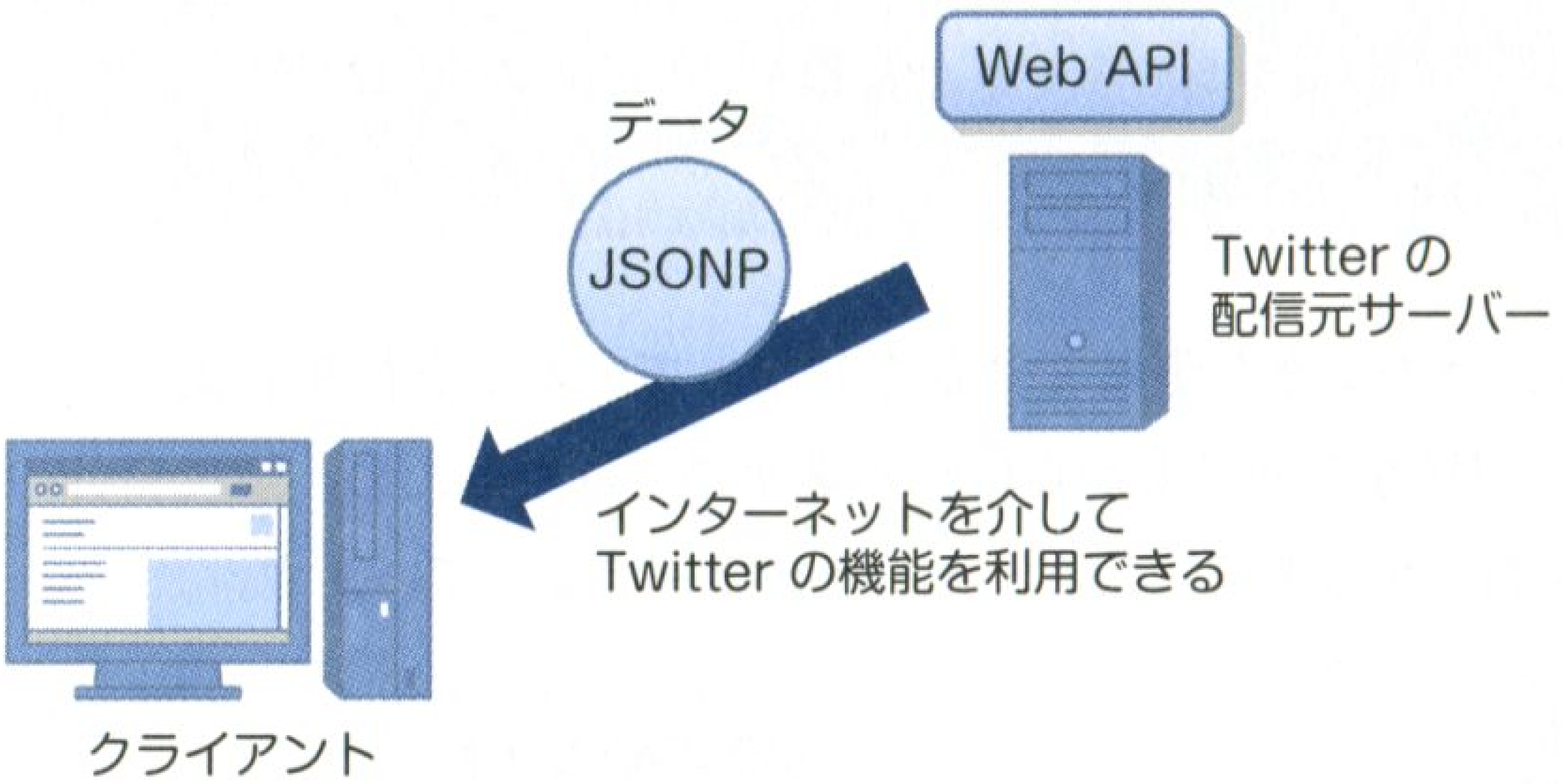
JSONPの場合、どのような手段で通信を行なっているかを簡単に説明しましょう。ドメインが異なるサイトからはデータが得られないため、逆に自分のサイトにデータを持ってきてもらおう、という仕組みになっています。



通常のデータ通信との違い

Web APIとは

Web APIとは、インターネットを介して利用することができる機能群のことです。GoogleやYahooなどのWebサービスの配信元が、サービスを活用するためのWeb APIを提供しています。ここでは、その中でも有名なTwitter APIを利用してみましょう。Twitter APIはJSONPに対応しています。リクエストパラメータとして検索キーワードを指定できるため、気になるツイートを取得できます。



Twitter API

JSONPの利用

JSONPを利用するには、JSONPを提供しているWebサイトに対してコールバック関数名を伝えます。コールバック関数の名前は、リクエストパラメータ（URLに付けて渡す引数）として相手に渡します。リクエストパラメータは、次のようにURLの末尾に「?」を挟んで指定します。パラメータが複数ある場合は&でつなげます。

リクエストパラメータの指定

```
URL?パラメータ1=値1&パラメータ2=値2
```

JSONPを利用する場合、コールバック関数は通常、次のように指定します。

コールバック関数の指定

```
callback=関数名
```

ただし、jQueryの場合は関数名を指定する必要がありません。次のようにcallbackパラメータの値として「?」を渡すと、getJSONメソッドで指定したコールバック関数に置き換えられます。

文法 jQueryによるJSONPの利用

```
$.getJSON("JSONPのURL?callback=?", コールバック関数);
```

※以下のソースコードは、本書の刊行後、TwitterAPIのバージョンアップにより動作しなくなりました。
2015年5月現在、代替コードを弊社Webサイトの本書のページ（追加情報）に掲載しております。

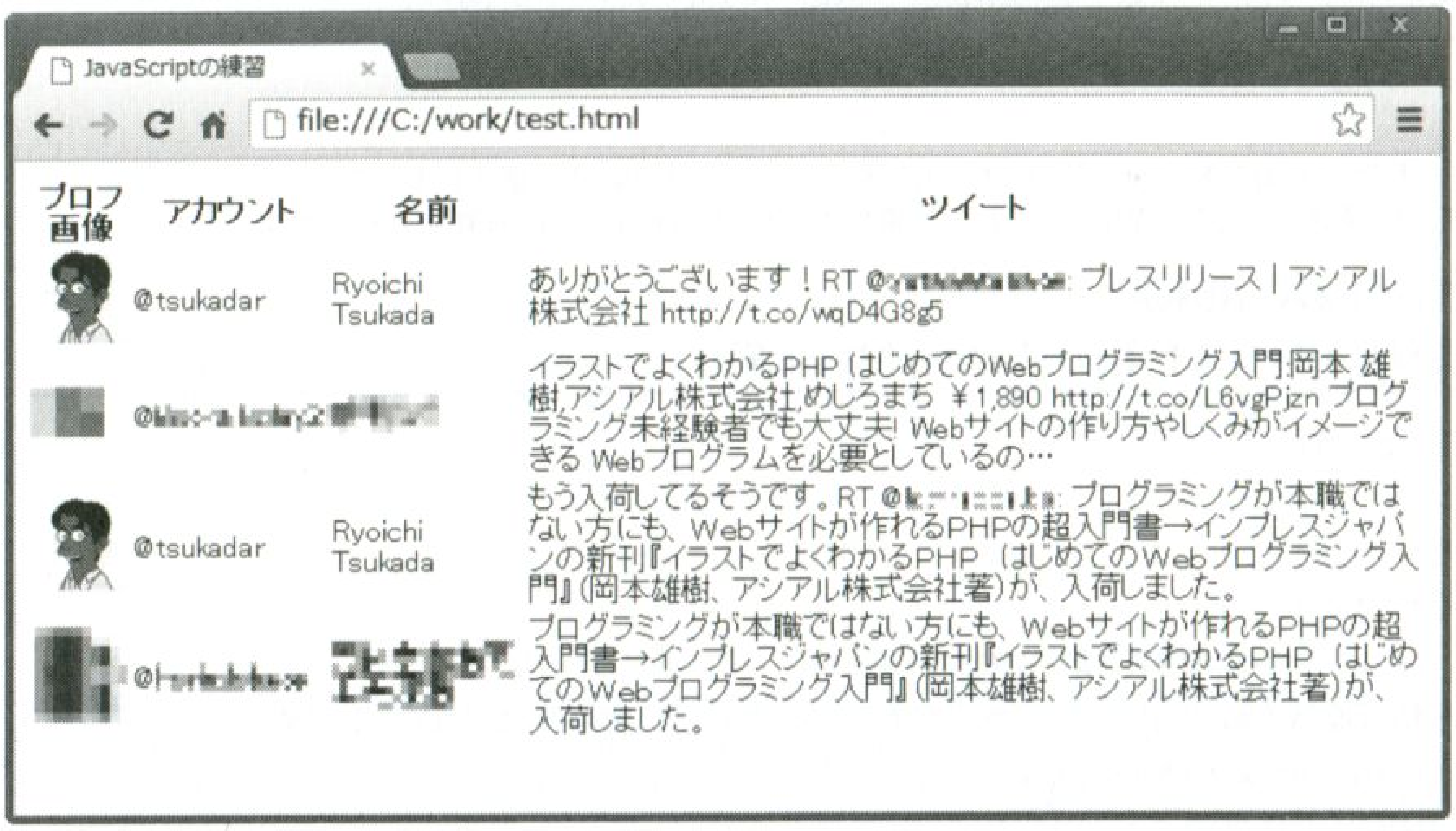
リスト13.3 TwitterAPIの利用

test.html （省略部分はリスト13.1のmain.htmlと同じ）

```
<!DOCTYPE html>
  (略)
  <script>
   $(function() {
     var q = encodeURIComponent("アシアル株式会社");
     $.getJSON("http://search.twitter.com/search.
json?callback=?&q=" + q,
       function(data) {
         for(var i=0; i<data.results.length; i++) {
           $("table").append(
             "<tr>" +
             "<td><img src='" + data.results[i].
profile_image_url + "'></td>" +
             "<td>@" + data.results[i].from_user +
             "</td>" +
             "<td>" + data.results[i].from_user_name +
"</td>" +
             "<td>" + data.results[i].text + "</td>" +
             "</tr>"
           );
         }
       }
     );
   });
  </script>
 </head>
 <body>
  <table>
```

```
    <tr>
      <th>プロフ画像</th>
      <th>アカウント</th>
      <th>名前</th>
      <th>ツイート</th>
    </tr>
  </table>
 </body>
</html>
```

■実行結果



とても便利なJSONPですが、利用する際はセキュリティに注意しなければなりません。JSONPを提供する側にもし悪意があった場合、コールバック関数の引数としてコンピュータに害をなすスクリプトを渡してくる可能性なども考えられます。外部から関数を利用されるというのは決して軽く考えて良いことではありません。まずはJSONPの提供者が信用できる相手かどうかを事前に確認し、信用できる場合のみ利用するようにしましょう。

Point!

- ▶ 同一ドメインからはHTMLやJSONをデータとして取得することができる。
- ▶ 外部ドメインからデータを受け取る必要がある場合は、JSONPを利用する。



この章の到達度チェック

Q1 同一ドメイン内のWebサイトからJavaScriptでデータを受け取るときに適したデータフォーマットはなんですか？

Q2 外部ドメインのサイトからJavaScriptでデータを受け取るときに適したデータフォーマットはなんですか？

Q3 以下は、data.jsonを取得するスクリプトです。空欄を埋めて完成させてください。

```
[      ]("data.json", function(data) { console.log(data); });
```

Q4 以下のJSONは誤った形式で記述されています。間違っているポイントをすべて挙げてください。

```
{ name : 'yamada', age : 25, hobby : 'swimming' }
```

Q5 異なるドメイン間でデータ通信を行なうことをなんと呼びますか？

Q6 getJSONメソッドでJSONPを取得するには、リクエストパラメータとしてなんというパラメータ名と値を渡せば良いですか？

➲解答は巻末に掲載

13

第14章

アニメーション

この章で学ぶこと

- 表示／非表示の切り替え
- 独自アニメーションの設定
- スライドショーの作成



● jQueryで手軽にアニメーションを作ろう！ ●



以前はWebページのアニメーションといえばFlashが主流でした。しかし簡単なアニメーションならjQueryでも十分実現でき、Flashよりも手軽でコストがかかりません。そのため、最近ではjQueryでアニメーションを制作するケースが増えています。

14

14 1 表示状態の切り替え

アニメーションとエフェクト

第12章では要素の表示／非表示を切り替えるためにCSSのdisplayプロパティを設定しましたが、jQueryには要素の表示状態を操作するメソッド群があらかじめ用意されています。各メソッドには引数としてアニメーション速度（要素の表示速度）を指定できるため、画像などを切り替える際にWebページに動きをつけることができます。つまり、Webページ上で簡単なアニメーションを実現できます。

各メソッドの違いは、エフェクトにあります。エフェクトとは、アニメーションの動き方のことです。本節では、3種類の異なるエフェクトを紹介します。

- 左上を基点に切り替える
- 上下にスライドする
- ぼんやり透けていく

左上を基点に切り替える ～show／hide／toggleメソッド

showメソッドで要素を表示し、hideメソッドで要素を非表示にします。toggleメソッドは、呼び出されるたびに要素の表示状態を交互に切り替えます。表示状態が変更されるまでの見た目の移り変わりを、アニメーションとして表示します。これらのメソッドは要素の左上を基点として動作します。

文法 showメソッド

```
要素.show(速度);
```

文法 hideメソッド

```
要素.hide(速度);
```

文法 toggleメソッド

```
要素.toggle(速度);
```

引数（速度）には、表示状態の切り替えが完了するまでにかかる時間を指定します。ミリ秒単位の数値か、以下の固定値のうちいずれかを指定します。

- "slow"………遅い（600ミリ秒）
- "normal"……普通（400ミリ秒）
- "fast" ………速い（200ミリ秒）

なお、引数を省略した場合は即座に表示状態の切り替えが行なわれます。

では、これらのメソッドを使って、背景色の表示状態を切り替える例を見てみましょう。

リスト14.1　表示状態の切り替え

HTML list14_1.html

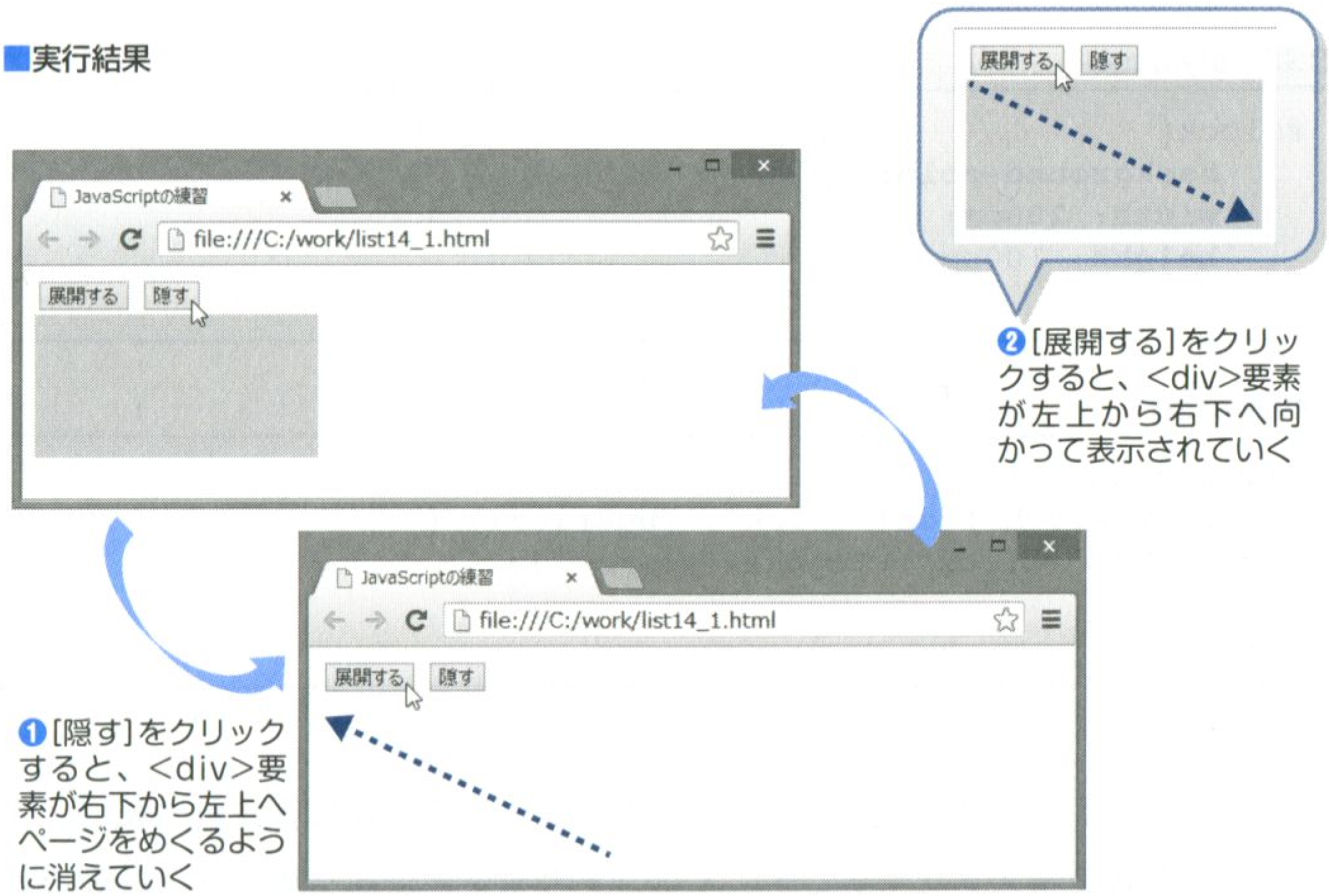
```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/ ↵
jquery/1.9.1/jquery.min.js"></script>
  <script>
   $(function() {
       $("#open").on("click", function() {
           $("#block").show("normal");
       });
       $("#close").on("click", function() {
           $("#block").hide("normal");
       });
   });
  </script>
  <link rel="stylesheet" type="text/css" href="style.css">
 </head>
 <body>
```

```
  <div>
   <button id="open">展開する</button>
   <button id="close">隠す</button>
  </div>
  <div id="block"></div>
 </body>
</html>
```

CSS style.css

```
#block{
    background-color: pink;
    width: 200px;
    height: 100px;
}
```

■実行結果



リスト14.1で［隠す］をクリックすると、ピンクの<div>要素が左上のほうに吸い込まれるようになくなっていき、最後には完全に消えてしまいます。逆に［展開する］をクリックすると、<div>要素が左上から右下へ伸びるように表示されていくのが確認できるでしょう。

> **Note** 以降のサンプルコードでは、<body>要素とJavaScript（リスト14.1の網掛け部分）を中心に掲載します。

続いて、これと同じ効果を1つのボタンで実現するスクリプトを見てみましょう。

リスト14.2　表示状態の切り替え

HTML　（<body>要素内のみ抜粋）

```
<div>
 <button id="switch">切り替え</button>
</div>
<div id="block"></div>
```

CSS　style.css

```
#block{
    background-color: pink;
    width: 200px;
    height: 100px;
}
```

JavaScript

```
$(function() {
    $("#switch").on("click", function() {
        $("#block").toggle("normal");
    });
});
```

リスト14.2は、［切り替え］ボタンがクリックされるたびに背景色の表示／非表示を切り替えるスクリプトです。イベントの発生元が同じ場合（この場合は［切り替え］ボタン）は、toggleメソッドを使ったほうがすっきりと記述できます。

上下にスライドする ～slideDown／slideUpメソッド

slideDownメソッドで要素を表示し、slideUpメソッドで要素を非表示にしま

す。実行すると、要素が上下にスライドするような動作をします。

引数に指定できる値はshow／hideメソッドと同様です。引数を省略した場合、速度は"normal"になります。

文法 slideDownメソッド

```
要素.slideDown(速度);
```

文法 slideUpメソッド

```
要素.slideUp(速度);
```

リスト14.3は、<div>要素を上下にスライドしながら表示の切り替えを行なうスクリプトです。

リスト14.3 上下にスライドするアニメーション

HTML （<body>要素内のみ抜粋）

```
<div>
 <button id="open">展開する</button>
 <button id="close">隠す</button>
</div>
<div id="block"></div>
```

CSS style.css

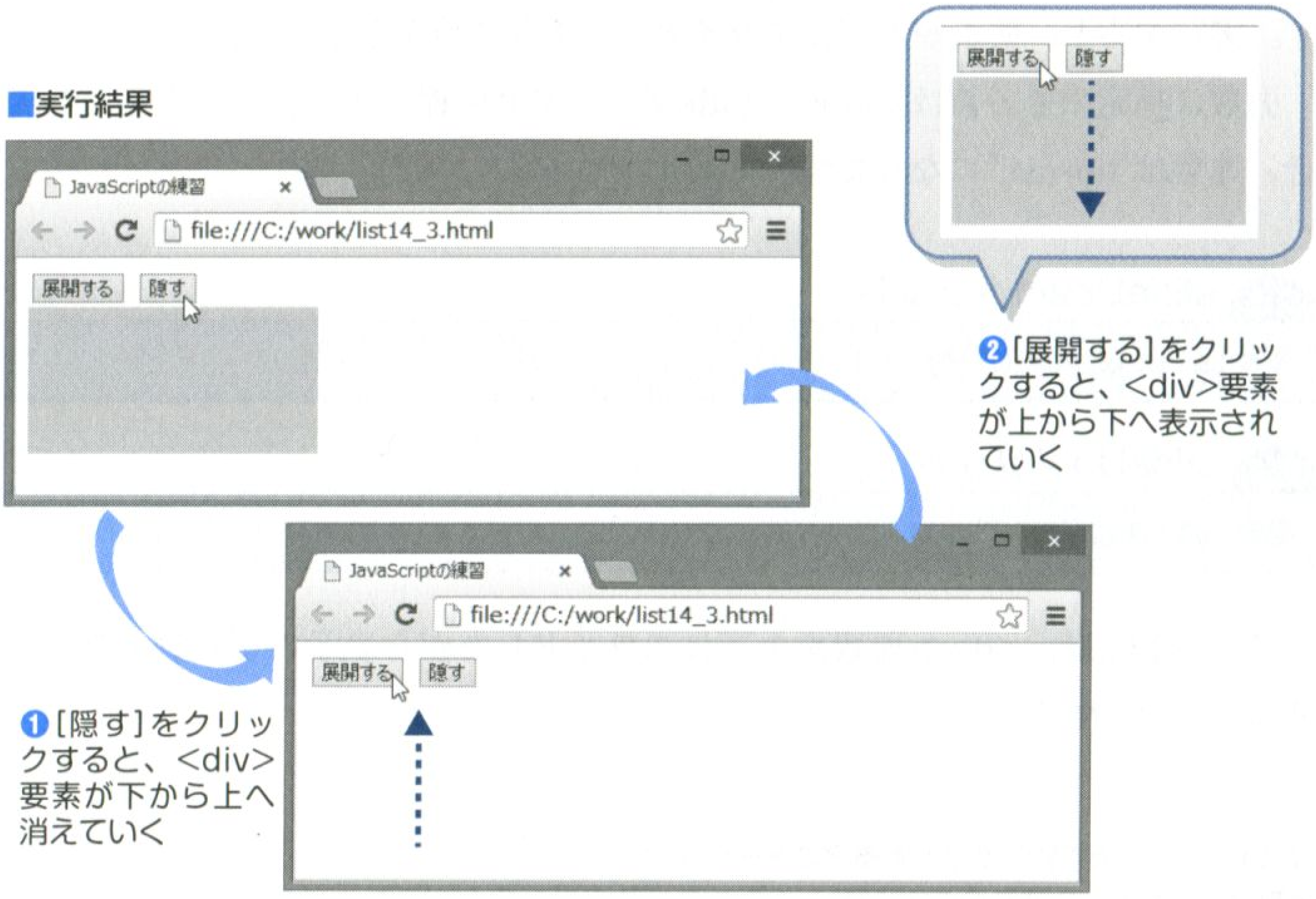
```
#block{
    background-color: pink;
    width: 200px;
    height: 100px;
}
```

JavaScript

```
$(function() {
    $("#open").on("click", function() {
        $("#block").slideDown("normal");
    });
    $("#close").on("click", function() {
        $("#block").slideUp("normal");
    });
});
```

14

■実行結果



ぼんやり透けていく ～fadeIn／fadeOutメソッド

fadeInメソッドで要素を表示し、fadeOutメソッドで要素を非表示にします。要素の位置やサイズは変わらず、透明度のみが変化します。実行すると、要素がそのままの位置でぼんやりと透けていく様子が確認できるでしょう。

引数に指定できる値はshow／hideメソッドと同様です。引数を省略した場合、速度は"normal"になります。

文法 fadeInメソッド

```
要素.fadeIn(速度);
```

文法 fadeOutメソッド

```
要素.fadeOut(速度);
```

画像の表示と非表示を行なう例を見てみましょう。

リスト14.4 透明度が変化するアニメーション

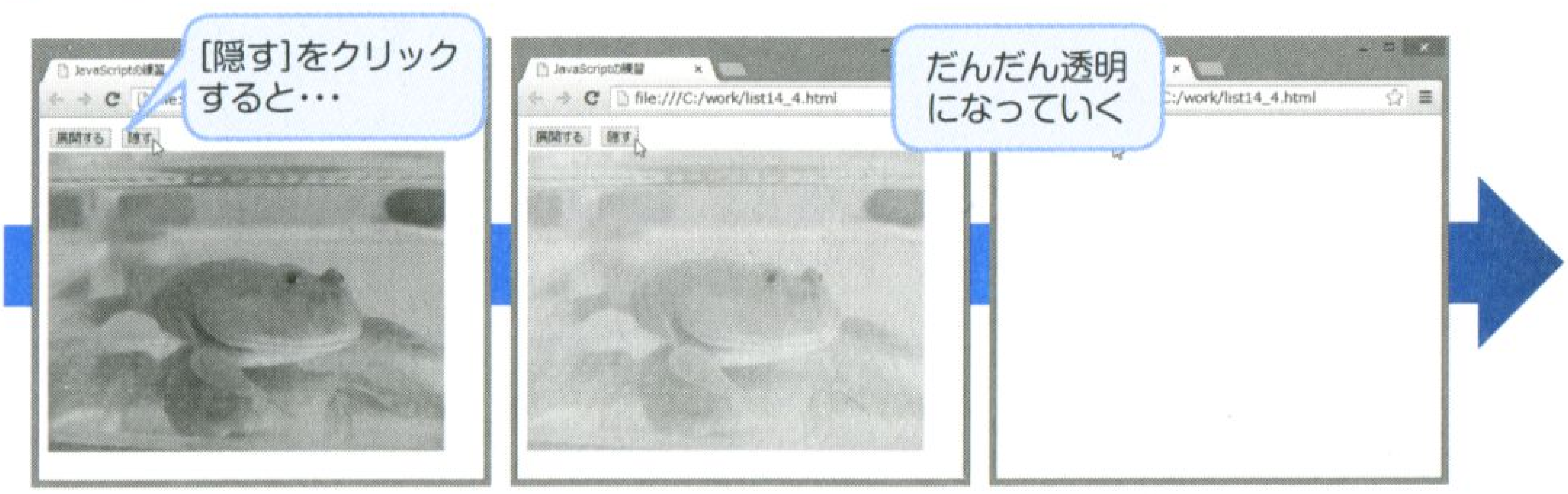
HTML （<body>要素内のみ抜粋）

```
<div>
 <button id="open">展開する</button>
 <button id="close">隠す</button>
</div>
<img src="images/frog1.jpg">
```

JavaScript

```
$(function() {
    $("#open").on("click", function() {
        $("img").fadeIn("normal");
    });
    $("#close").on("click", function() {
        $("img").fadeOut("normal");
    });
});
```

■実行結果



Point!

▶ 要素の表示状態を切り替えるときにアニメーションを設定することができる。



Check Test 1

Q1 実行するたびに<img>要素の表示状態を切り替えるには、空欄になにを記述すれば良いですか？

```
$("img").[        ]();
```

Q2 <img>要素を1秒かけて徐々に薄くするには、空欄になにを記述すれば良いですか？

```
$("img").fadeOut([        ]);
```

➲解答は巻末に掲載

14-2 独自アニメーションの作成

animateメソッドによるアニメーション

独自アニメーションを設定するメソッドを使えば、エフェクトを自由に設定できます。

animateメソッドは、作成者が自由な動きを指定できるメソッドです。組み込みエフェクトとして用意されていない動きを与えたい場合に利用します。

文法 animateメソッド

```
要素.animate( { CSSプロパティ群 } , 速度, イージング, コールバック関数);
```

第2～4引数は省略可能

animateメソッドは、現在の要素の状態から、第1引数で指定したスタイルが適用されるまでの変化をアニメーションとして表示します。変化する速度は第2引数で指定します。

第3引数のイージングは、アニメーションの実行速度に緩急をつけるものです。次の2つのうちどちらかを指定します。

- "linear" ……一定速度で動く
- "swing" ……出だしは遅く、後半は速め

第4引数には、アニメーションが完了したタイミングで実行する関数を指定します。第2引数以降は省略が可能です。

簡単なアニメーションの例を見てみましょう。

リスト14.5　独自アニメーションの実行

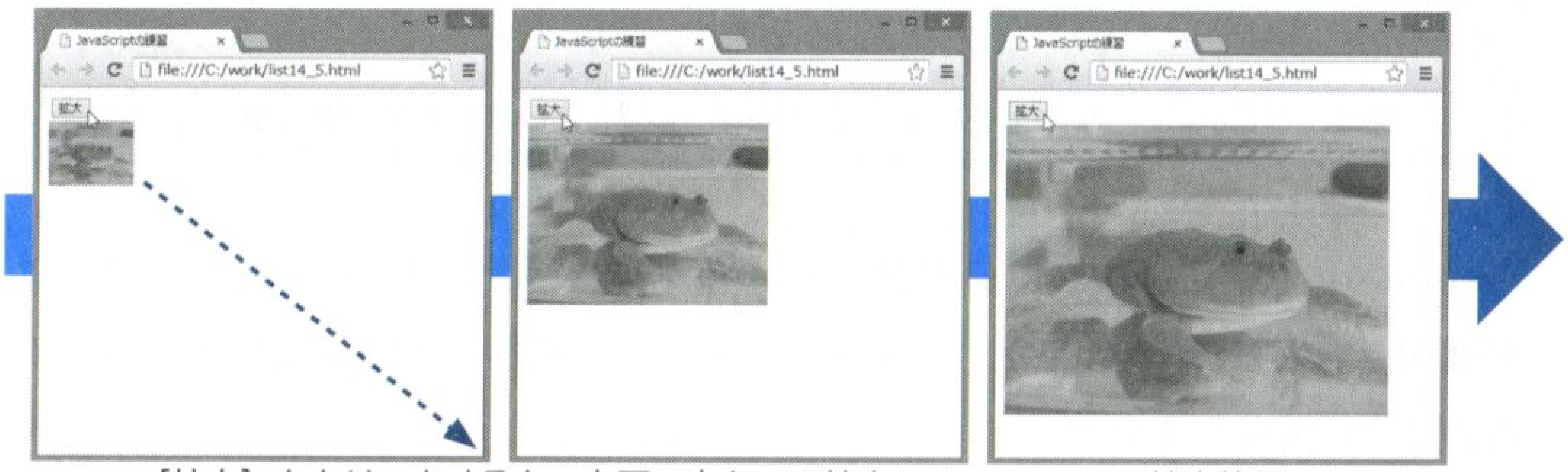
HTML　(<body>要素内のみ抜粋)

```
<div>
 <button id="large">拡大</button>
</div>
<img src="images/frog1.jpg" id="photo" width="20%" height="20%">
```

JavaScript

```
$(function() {
    $("#large").on("click", function() {
        $("#photo").animate({
            "width" : "90%",
            "height" : "90%"
        }, 2000, "linear");
    });
});
```

■実行結果



［拡大］をクリックすると、右下へ向かって拡大していく

拡大終了

リスト14.5は、画像を拡大するアニメーションです。まず、画像の初期サイズはウィンドウサイズに対して20%に設定されています。

```
<img src="images/frog1.jpg" id="photo" width="20%" height="20%">
```

［拡大］ボタンクリック時の処理で、画像に対してアニメーションを設定しています。第1引数は、要素に適用するスタイルです。width、heightともに90%を指定しています。初期サイズからこのサイズになるまで変化していく様子が、2秒間のアニメーションとして表示されます。

```
$("#photo").animate({
    "width" : "90%",
    "height" : "90%"
}, 2000, "linear");
```

注意 指定できるCSSプロパティは値が数値型のものだけ

文字色や背景色などの色情報を値として持つCSSプロパティをanimateメソッドで変化させることはできません。

animateメソッドに対するメソッドチェーン

animateメソッドをメソッドチェーンでつなげることで、アニメーションを段階的に実行できます。

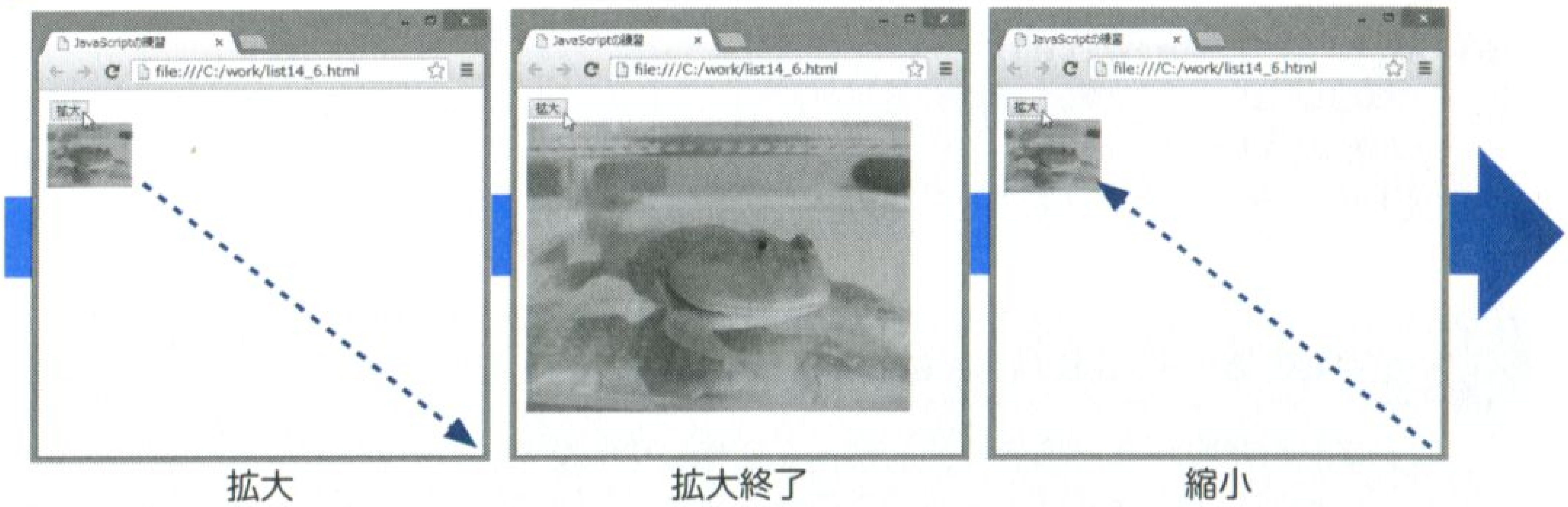
リスト14.5のスクリプトに以下の記述を追記すると、一度大きくなったあと、再び元の大きさに戻るアニメーションを作ることができます。

リスト14.6 animateメソッドの段階的な実行

```
$(function() {
    $("#large").on("click", function() {
        $("#photo").animate({
            "width" : "90%",
            "height" : "90%"
        }, 2000, "linear").animate({
            "width" : "20%",
            "height" : "20%"
        }, 2000, "linear");
    });
});
```

14

■実行結果



スライドショーの作成

それでは、animateメソッドを応用して、画像が左右に流れていくスライドショーを自作してみましょう。

リスト14.7　スライドショー

HTML　index.html

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
   <script src="http://ajax.googleapis.com/ajax/libs/
jquery/1.9.1/jquery.min.js"></script>
   <script src="slideshow.js"></script>
   <link rel="stylesheet" type="text/css" href="style.css">
 </head>
 <body>
  <div id="photoFrame">
   <div id="imageArea"></div>
  </div>
  <div>
   <button id="prev">前へ</button>
   <button id="next">次へ</button>
  </div>
 </body>
</html>
```

CSS style.css

```
* {
    margin: 0;
    padding: 0;
}
body {
    padding: 30px;
}
#photoFrame {
    position: relative;
    width: 200px;
    height: 150px;
    overflow: hidden;
}
#imageArea {
    position: absolute;
    left: 0px;
    width: 1000px;
    height: 150px;
    white-space: nowrap;
}
img {
    width: 200px;
    height: 150px;
}
```

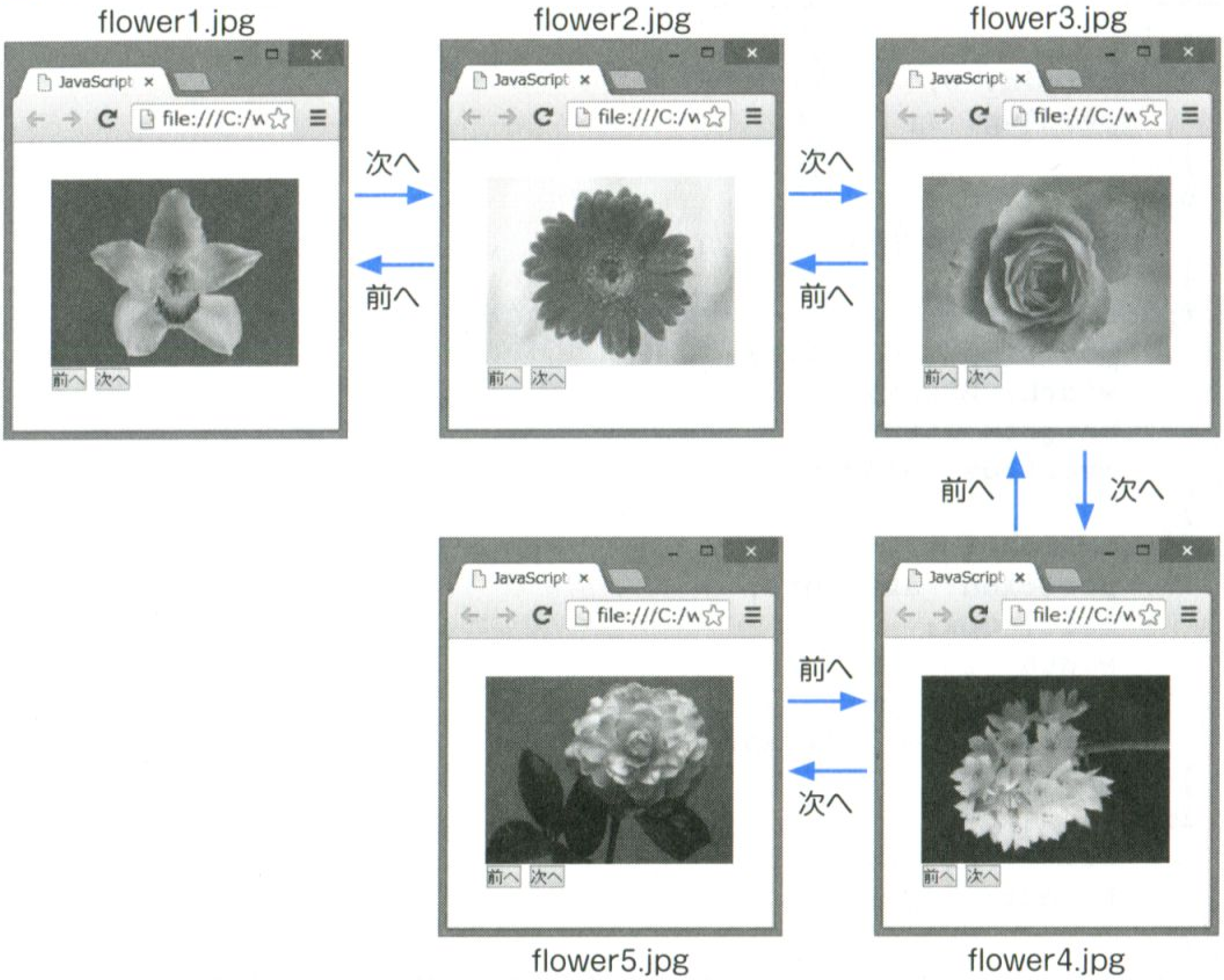
JavaScript slideshow.js

```
$(function() {
  var imageArea = $("#imageArea");
  for(var i=1; i<=5; i++) {                          ❶
    imageArea.append("<img src='images/flower" + i + ".jpg'>");
  }

  $("#next").on("click", function() {                ❷
    if(parseInt(imageArea.css("left")) > -800) {
      imageArea.animate({"left" : "-=200px" }, "fast", "linear");
    }
  });

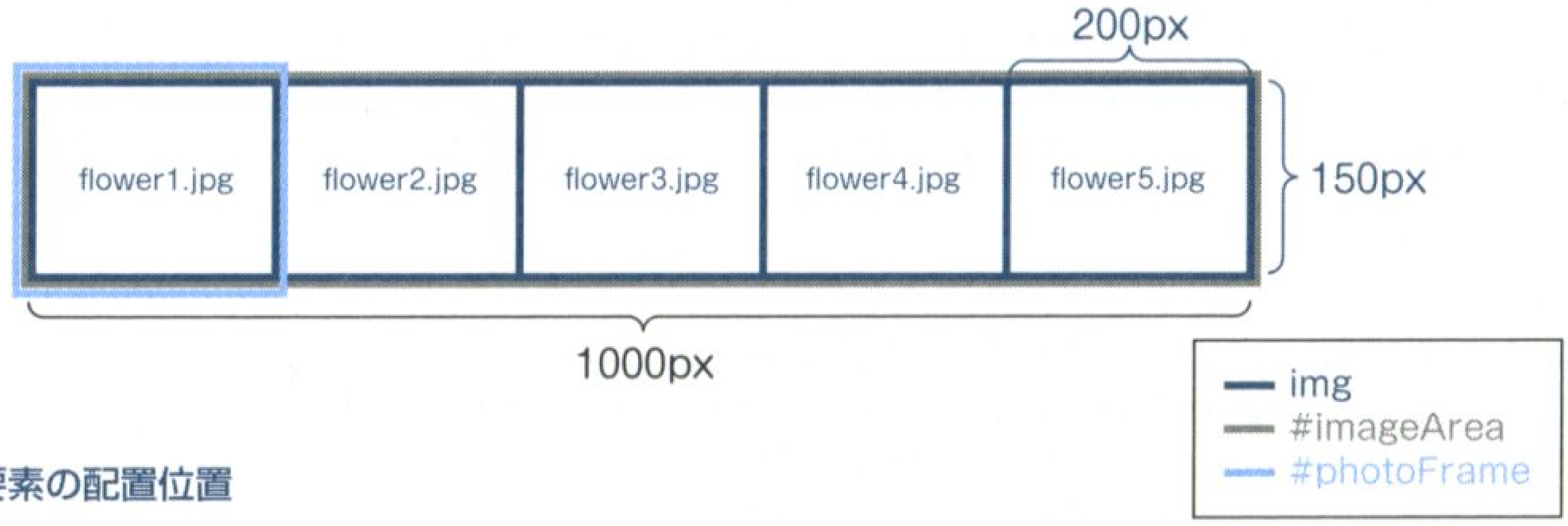
  $("#prev").on("click", function() {
    if(parseInt(imageArea.css("left")) < 0) {
      imageArea.animate({"left" : "+=200px" }, "fast", "linear");
    }
  });
});
```

■実行結果



スライドショーの解説～HTMLとCSS

まずはJavaScriptの❶の部分を見てください。#imageArea要素の子要素に<img>要素を5件追加しています。その結果、#photoFrame要素以下のHTMLタグは次のように配置されます。



要素の配置位置

各要素に設定されたスタイルを詳しく見てみましょう。

```
img {
    width: 200px;
    height: 150px;
}
```

<img>要素には、幅200px／高さ150pxが設定されています。

```
#imageArea {
    position: absolute;
    left: 0px;
    width: 1000px;
    height: 150px;
    white-space: nowrap;
}
```

#imageArea要素には、幅1000px／高さ150pxが設定されています。幅が1000pxなのは、幅200pxの<img>要素を子要素として5つ持っているためです。

positionプロパティは、要素の配置方法を設定するプロパティです。親要素の値にrelative、子要素の値にabsoluteを指定すると、要素を配置する座標を親要素から見た相対位置で指定することができるようになります。leftプロパティに0pxが設定されていますが、これは親要素である#photoFrame要素の左端の座標とまったく同じ位置に配置されることを示します。仮にleftプロパティを10pxに変更すると、#photoFrame要素の左端から右に10pxずれた位置に配置されます。

white-spaceプロパティは、改行の表示方法を設定するプロパティです。値にnowrapを指定すると、要素内で折り返さなくなります。子要素が一列に連なるので、横に長い形になります。

```
#photoFrame {
    position: relative;
    width: 200px;
    height: 150px;
    overflow: hidden;
}
```

14

#photoFrame要素には、1件分の<img>要素と同じ幅と高さが設定されています。

overflowプロパティは、自身の表示領域からはみ出た子要素をどのように表示するか設定するプロパティです。値にhiddenを指定すると、はみ出た部分は表示されなくなります。よって、次図に示すとおり、#photoFrame要素の領域の外側は表示されません。実際は子孫要素として<img>要素が5件存在していますが、画面上に見えているのは先頭の1件のみになります。



要素の表示状態

◇ ボタンクリック時の動作

ここまでが前提です。それではいよいよ、[次へ]ボタンがクリックされたときの処理を見てみましょう。リスト14.7の❷の部分です。

```
$("#next").on("click", function() {
  if(parseInt(imageArea.css("left")) > -800) {      ←Ⓐ
    imageArea.animate({ ⏎                           ←Ⓑ
"left" : "-=200px" }, "fast","linear");             ←Ⓑ
  }
});
```

2行目のⒶの処理ですが、まず#imageArea要素のCSSプロパティleftに設定されている値を取得しています。初期状態ではleftには0pxが設定されています。

```
if(parseInt(imageArea.css("left")) > -800) {
            "0px"
```

次に、取得したleftプロパティ値を数値型に変換しています。parseInt関数は先頭の数字部分のみを抜き出すので、数値型の「0」が返ります。

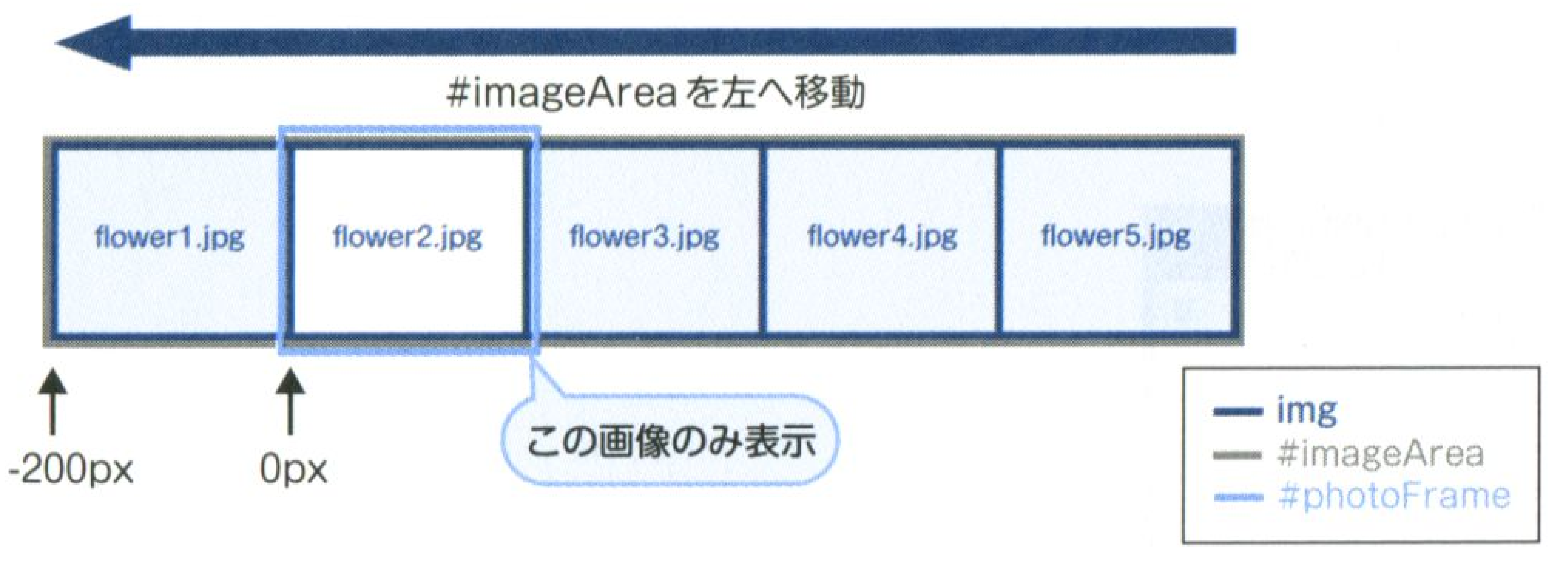
```
if(parseInt(imageArea.css("left")) > -800) {
   0
```

こうして求められた数値が-800よりも大きい場合はif文のブロック内を実行します。「-800」という数値の意味については後述します。

続いてⒷでは、アニメーションを実行しています。プロパティ値の指定の仕方が少し特殊ですが、このように複合代入演算子を指定すると、現在のプロパティ値に対して加減算を行なうことができます。初回実行時は0pxから200pxを引いた結果、-200pxがleftプロパティとして設定されます。

```
imageArea.animate({ "left" : "-=200px" }, "fast", "linear");
```

#imageArea要素のleftプロパティを0pxから-200pxに変化させると、要素の配置位置が次図のように変化します。

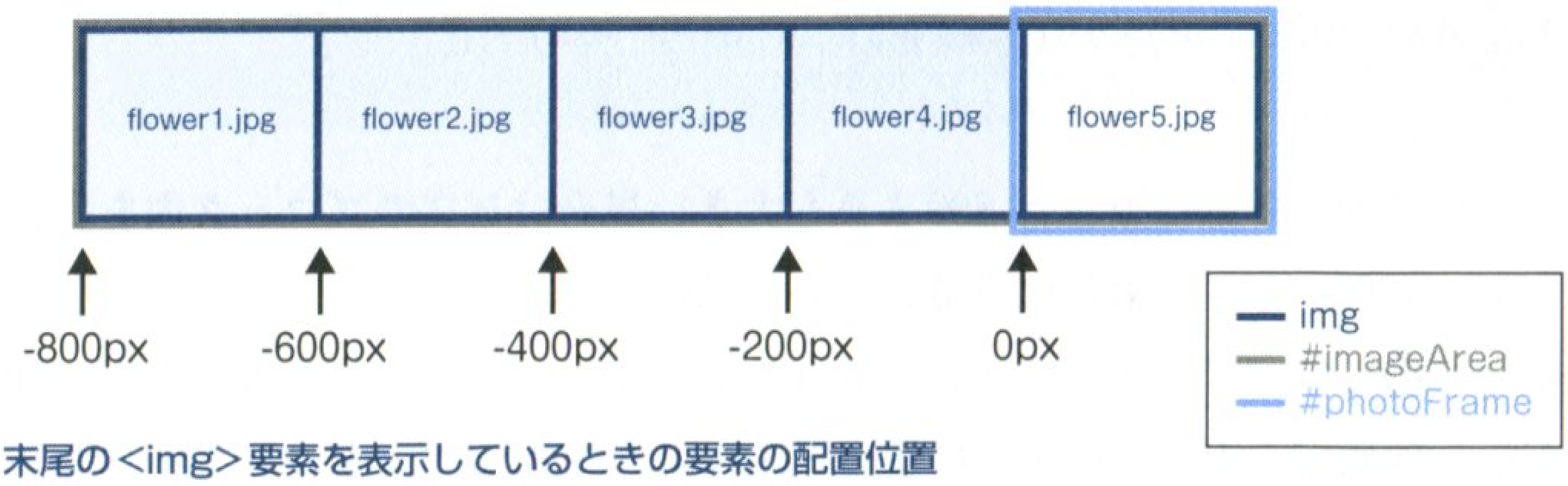


[次へ] ボタンを1回クリックした後の要素の配置位置

#photoFrame要素の位置は変わっていませんが、#imageArea要素の位置が左に200px分ずれたので、2つ目の<img>要素が画面に表示されるようになりました。leftプロパティを変更することで、あたかも画像が右から左へ流れていったように見せているのです。

[次へ] を繰り返して行くと、4回目で末尾の<img>要素にたどりつきます。この状態で [次へ] がクリックされても、次の要素は存在しないのでアニメーションを実行するわけにはいきません。そこで、leftプロパティの値が-800以下になった場合は処理を実行しないように、if文で条件を指定しておいたのです。

14

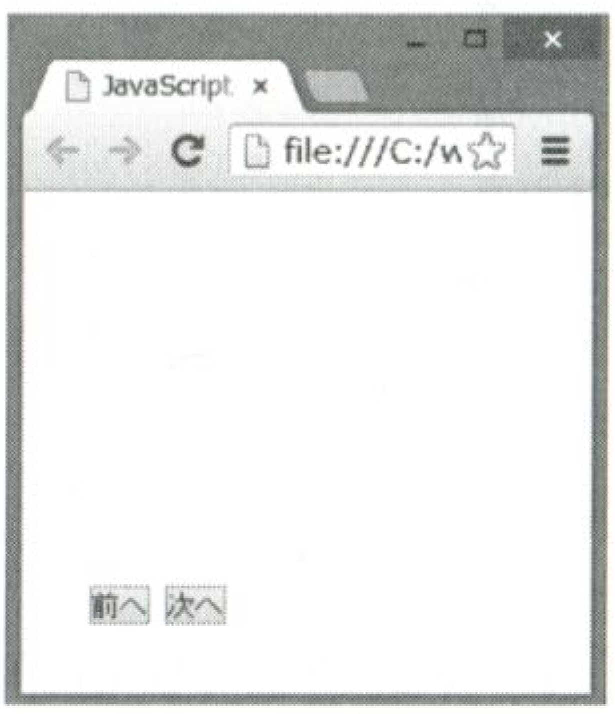


末尾の<img>要素を表示しているときの要素の配置位置

［前へ］がクリックされたときはleftプロパティに数値を加算して、逆方向に#imageArea要素を移動させます。このような仕組みで画像が左右に移動するスライドショーを実現しています。

スクリプトのバグ～キューとは

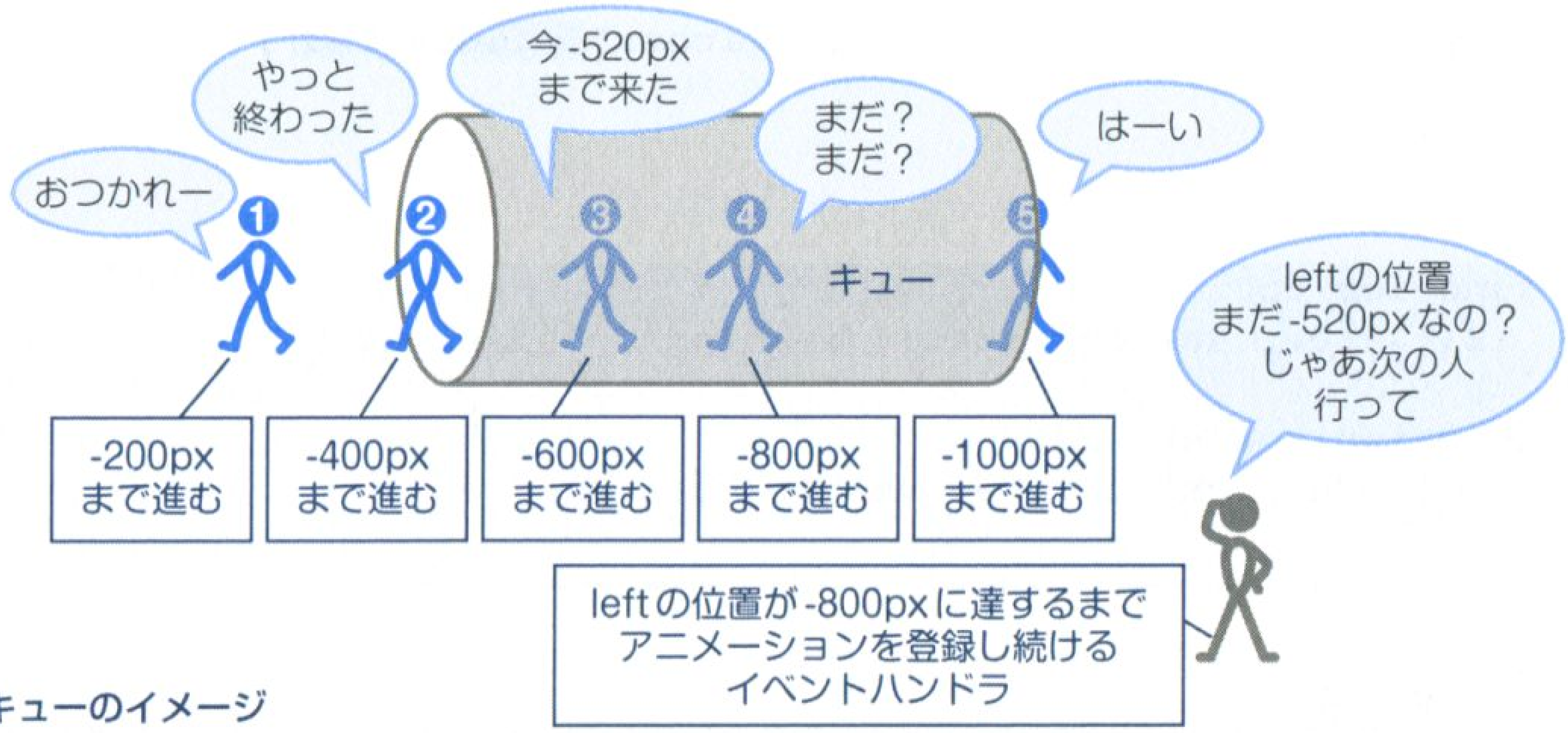
ところで、このスクリプトには実はバグ（不具合）があります。［次へ］ボタン（あるいは［前へ］ボタン）をすばやく何度もクリックすると、末端の要素を通りすぎてしまうことがあります。その場合、空のWebページが表示されるはずです。



空のWebページ

なぜこのような現象が起こるかというと、実はアニメーションは即座に実行されるわけではないのです。animateメソッドを呼び出すと、アニメーションは「キュー」という、処理を溜めておくトンネルのような場所に登録されます。キューに登録されたアニメーションは先頭から順に実行されていきますが、キューが混み合うとどんどんアニメーションの実行が遅れていきます。その状態でボタンのクリックイベントが実行されると、#imageArea要素のleftプロパティはまだ前回到達する予定だった位置にたどりついていないことがあり

ます。そのため、限界値の-800pxを超えてしまう現象が発生するのです。



キューのイメージ

そこで、キューに順番待ちのアニメーションが溜まらないようにするために、animateメソッドを呼び出してからアニメーションが完了するまでの間は、ボタンをクリックできないようにしてしまいましょう。リスト14.7のanimateメソッドにコールバック関数を追記してみてください（リスト14.8）。animateメソッドを呼ぶ直前で両ボタンにdisabled属性を設定し、アニメーション完了後に実行されるコールバック関数でdisabled属性を削除しています。

リスト14.8　コールバック関数の利用（slideshow.js）

```
$(function() {
  var imageArea = $("#imageArea");
  for(var i=1; i<=5; i++) {
    imageArea.append("<img src='images/flower" + i +".jpg'>");
  }

  $("#next").on("click", function() {
    if(parseInt(imageArea.css("left")) > -800) {
      toDisabled();                              ❶ボタン無効化
      imageArea.animate( ⏎
{"left" : "-=200px"},"fast", "linear", toEnabled);
    }
  });                      ❷アニメーションの実行      ❸ボタン有効化

  $("#prev").on("click", function() {
    if(parseInt(imageArea.css("left")) < 0)
```

```
      toDisabled();
      imageArea.animate( ↵
{"left" : "+=200px"},"fast", "linear", toEnabled);
    }
  });
});

function toDisabled() {
  $("#prev, #next").attr("disabled", "disabled");
}
function toEnabled() {
  $("#prev, #next").removeAttr("disabled");
}
```

Point!

- animateメソッドで独自のアニメーションを作ることができる。
- メソッドチェーンを利用すればアニメーションの連鎖を起こすことができる。



この章の到達度チェック

Q1 以下は、<img>要素の幅を400px、高さ250pxに変更するアニメーションを実行するスクリプトです。空欄を埋めて完成させてください。

```
$("img").animate({[          ]});
```

Q2 イージングの"linear"とはどういう意味ですか？

Q3 以下の<img>要素に対してスクリプトを実行すると、最終的に幅は何pxになりますか？

```
<img src="images/flower1.jpg" width="200px">
```

スクリプト

```
$("img").animate({ "width" : "+=100px" });
```

Q4 以下は、<div>要素にアニメーションを設定して高さを徐々に伸ばした後、フォントサイズも徐々に大きくするスクリプトです。空欄を埋めて完成させてください。

14

```
$("div").animate({ "height" : "300px" }, 1000).[      ] ⏎
({ "font-size" : "50px" }, 1000);
```

Q5 以下の<img>要素に対してスクリプトを実行すると、どのようなアニメーションが表示されますか？

```
<img src="images/flower1.jpg" width="200px">
```

スクリプト

```
$(function() {
    $("img").hide().animate({"width" : "400px"},1000,"swing");
});
```

Q6 以下の<img>要素に対してスクリプトを実行すると、どのようなアニメーションが表示されますか？

```
<img src="images/flower1.jpg" width="200px">
```

スクリプト

```
$(function() {
    $("img").animate( ⏎
{"width" : "400px"},1000,"linear",toPassive);
});
function toPassive(){
    $("img").hide();
}
```

➲解答は巻末に掲載

次のステップへ

あなたの手を引き、支えることができるのはここまでです。しかし心配しなくても大丈夫です。気づいていないかもしれませんが、あなたはもう初心者を脱却しているはずです。ここから先はあなた一人でも歩いて行くことができるでしょう。

JavaScriptやjQueryには、本書で紹介しきれなかった機能がまだまだたくさんあります。幸い、Web上にはリファレンスが山のようにありますから、ちょっと検索すれば情報はいくらでも出てきます。今の時代は必要な情報を容易に入手することができるのですから、その恩恵をたっぷりと受けて、貪欲に新しい知識を取り入れていきましょう。

「もっとJavaScriptを極めたい！」そんなあなたは、

- **まずはなんでも良いので、スクリプトを1つ作ってみる**

ことです。特になにも思いつかない場合は、既存のWebサイトやWebサービスの模倣でかまいません。とにかくスクリプトを完成させることが重要なのです。やってみればわかると思いますが、まず間違いなくうまくいきません。しかし落ち込む必要はまったくありません。スクリプトがうまく動かないときこそ、スキルアップするチャンスなのです。問題解決のために試行錯誤を繰り返すうちに、自然とスクリプトの書き方が身についていくものです。そうやって続けていれば、いずれ、見違えるように成長した自分に気づくことでしょう。

JavaScriptのサンプル集

本書の公式Webページでは、Webサイトの制作でよく使われている小技的なJavaScriptのサンプル集をPDFで公開しています。

http://www.shoeisha.co.jp/book/surasura/

このサンプル集では、以下のようなJavaScriptを取り上げています。これらのスクリプトの書き方を一通り理解できれば、JavaScriptの初心者を卒業したと言えるでしょう。本書の学習の総仕上げとして、これらのスクリプトの作成に取り組んでみてください。

- 実用的な入力チェックを行なう
- パスワードの入力チェックを即時実行する
- サムネイル画像のサイズを切り替える
- JavaScriptが有効かどうかをチェックする
- テーブルの特定行を強調する
- 入力中のテキストボックスの背景色を変える
- ローディング画面を表示する
- 長い文章を省略し、「続きを読む」リンクを表示する
- 金額をカンマ区切りで表示する
- 開閉式のメニューを作る
- 自動的に隠れるナビゲーションメニューを作る
- ページトップへ戻るボタンを表示する
- スクロールで画像がめくれる動きを表現する
- ドアが開くようなアニメーションを表現する
- アバター機能を作る
- タブメニューを作る
- サムネイル画像付きのスライドショーを作る
- ループするスライドショーを作る
- 一定時間おきに自動で動くスライドショーを作る

Appendix

付 録

プログラムがうまく動かないときの対処法～デバッグ

A 1 プログラムのバグを取り除こう

JavaScriptで記述したスクリプト（プログラム）がうまく動かないときは、プログラムの中に**バグ**が潜んでいないかどうかを疑ってください。バグとは、プログラムに含まれる誤りや不具合のことです。そして、プログラムからバグを取り除く作業のことを**デバッグ**と呼びます。

各種ブラウザには、デバッグを行なうためのツールが用意されています。本書では、Google ChromeのJavaScriptコンソールを使った利用方法を紹介します。

バグの要因

バグの原因には、大きく分けて2つの種類があります。

- プログラムの記述ミス（構文エラー）
- 意図したとおりに動かない

以降で、これらについて見ていきましょう。



2 スクリプトの記述ミス（構文エラー）の場合

まずは、1つ目の記述ミスによる構文エラー（SyntaxError）です。これは構文エラー、つまり文法の書き間違いです。

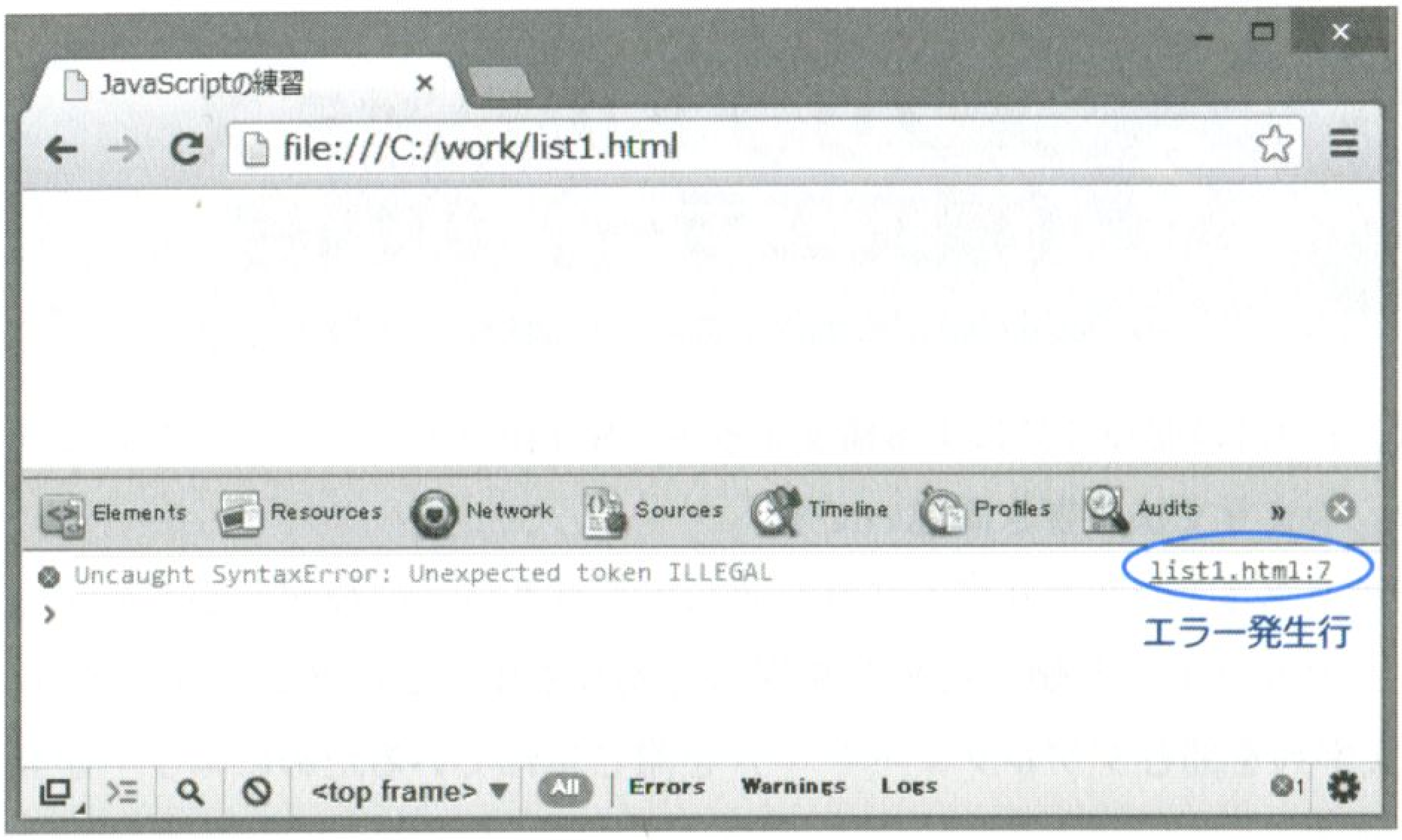
さっそくリスト1のスクリプトに目を通してください。行数がわかりやすいように、スクリプトの左側に行番号を振ってあります。このスクリプトでは、7行目で文字列を囲むダブルクォート（"）を閉じ忘れているためにエラーが発生しています。

リスト1　記述ミスがあるスクリプト

```
 1: <!DOCTYPE html>
 2: <html lang="ja">
 3:  <head>
 4:   <meta charset="utf-8">
 5:   <title>JavaScriptの練習</title>
 6:   <script>
 7:     alert("こんにちは);
 8:   </script>
 9:  </head>
10:  <body>
11:  </body>
12: </html>
```

ここに"の記述が漏れている

このような場合、原因の特定は比較的簡単です。エラーが発生しているときには、JavaScriptコンソールにエラーの内容と発生箇所が表示されます。



JavaScript コンソールにはエラー発生行が表示される

> **Note** リスト1のHTMLファイルを作成し、ブラウザで表示すると空白ページが表示されます。空白ページが表示された状態で、キーボードの［F12］キーを押すとJavaScriptコンソールが表示されます（もう一度［F12］キーを押すとJavaScriptコンソールが閉じます）。

このように、「SyntaxError」と表示された場合は、該当行にスクリプトの記述ミスがないか確認しましょう。

A 3 意図したとおりに動かない場合

さて、厄介なのは、バグの要因の2つ目です。これは、「文法は間違っていないのでエラーは発生しないが、スクリプトが意図したとおりに動かない」という場合です。このようなときは、スクリプトの目視確認を行なって問題箇所を特定するしかありませんが、そう簡単には見つからないことでしょう。そこで役立つのが、JavaScriptコンソールの「ブレークポイント」と「ステップ実行」機能です。

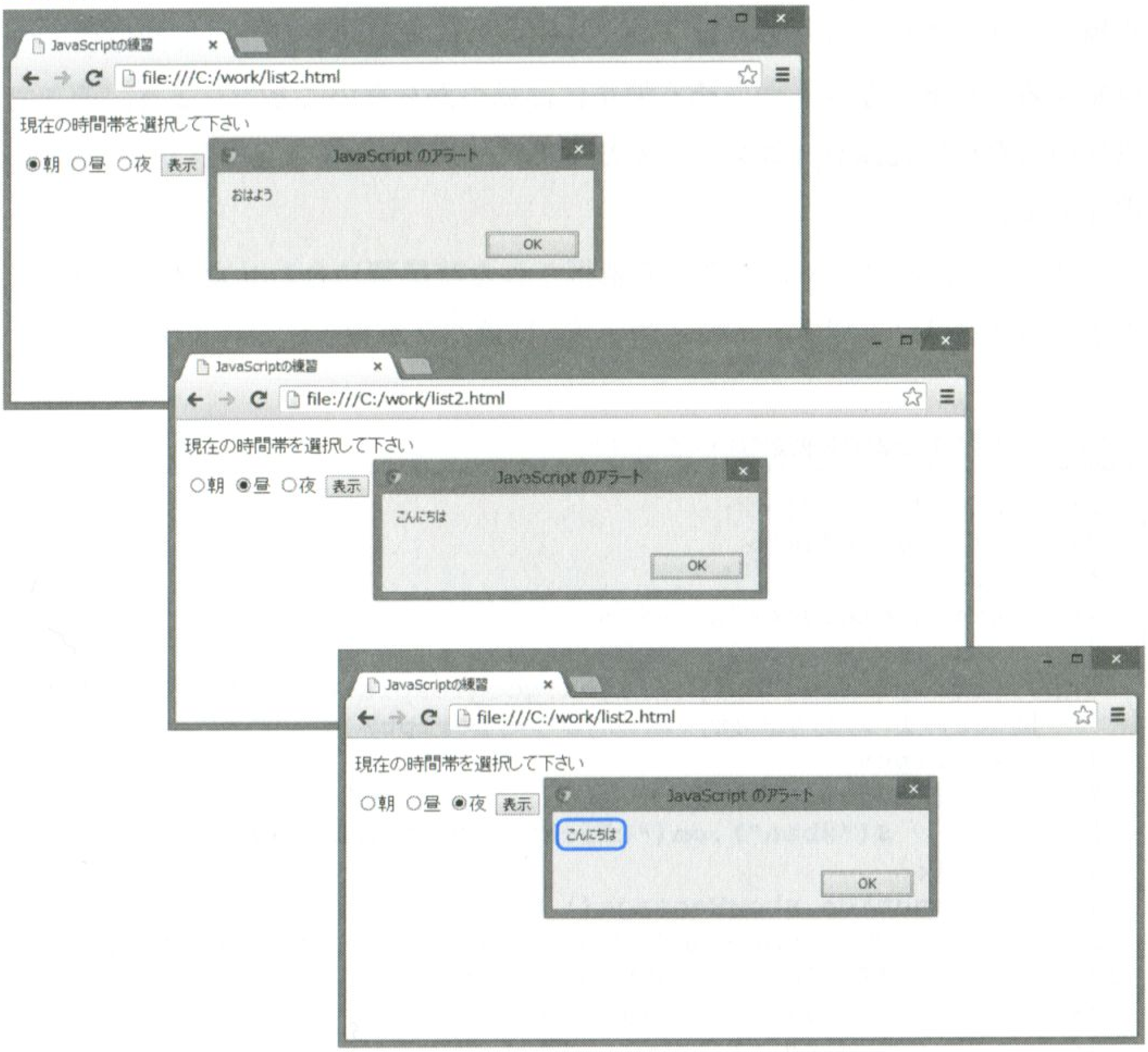
リスト2は、オプションボタンで選択された時間帯に合わせたメッセージを表示するスクリプトです。このスクリプトにはバグが含まれています。

リスト2　意図したとおりに動かないスクリプト

```
<!DOCTYPE html>
<html lang="ja">
 <head>
  <meta charset="utf-8">
  <title>JavaScriptの練習</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
  <script>
     $(function() {
        $("#btn").on("click", showMessage);
    });
    function showMessage() {
        var time = $("input[name='time']:checked").val();
        if(time == "朝") {
            alert("おはよう");
        } else if(time = "昼") {
            alert("こんにちは");
        } else if(time == "夜") {
            alert("こんばんは");
        } else {
            alert("いずれかの時間帯を選択して下さい");
        }
    }
  </script>
 </head>
```

```
  <body>
    <p>現在の時間帯を選択して下さい</p>
    <input type="radio" name="time" value="朝">朝
    <input type="radio" name="time" value="昼">昼
    <input type="radio" name="time" value="夜">夜
    <input type="button" id="btn" value="表示">
  </body>
</html>
```

■実行結果



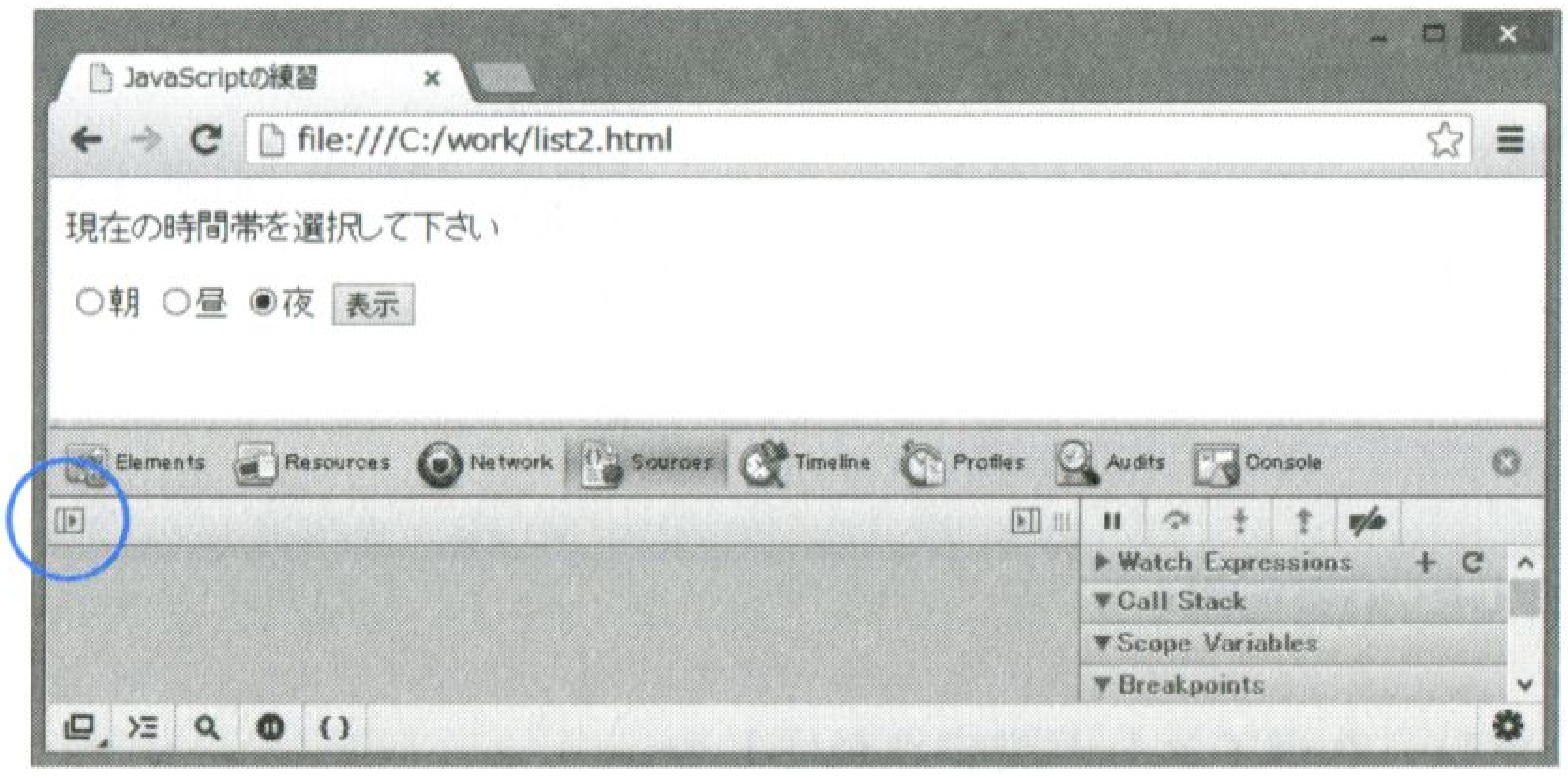
このスクリプトを実行すると、「朝」と「昼」を選択した場合は正しく動くにもかかわらず、「夜」を選択した場合には「こんにちは」というメッセージが表示されてしまいます。原因をJavaScriptコンソールで特定してみましょう。

ブレークポイントの設定

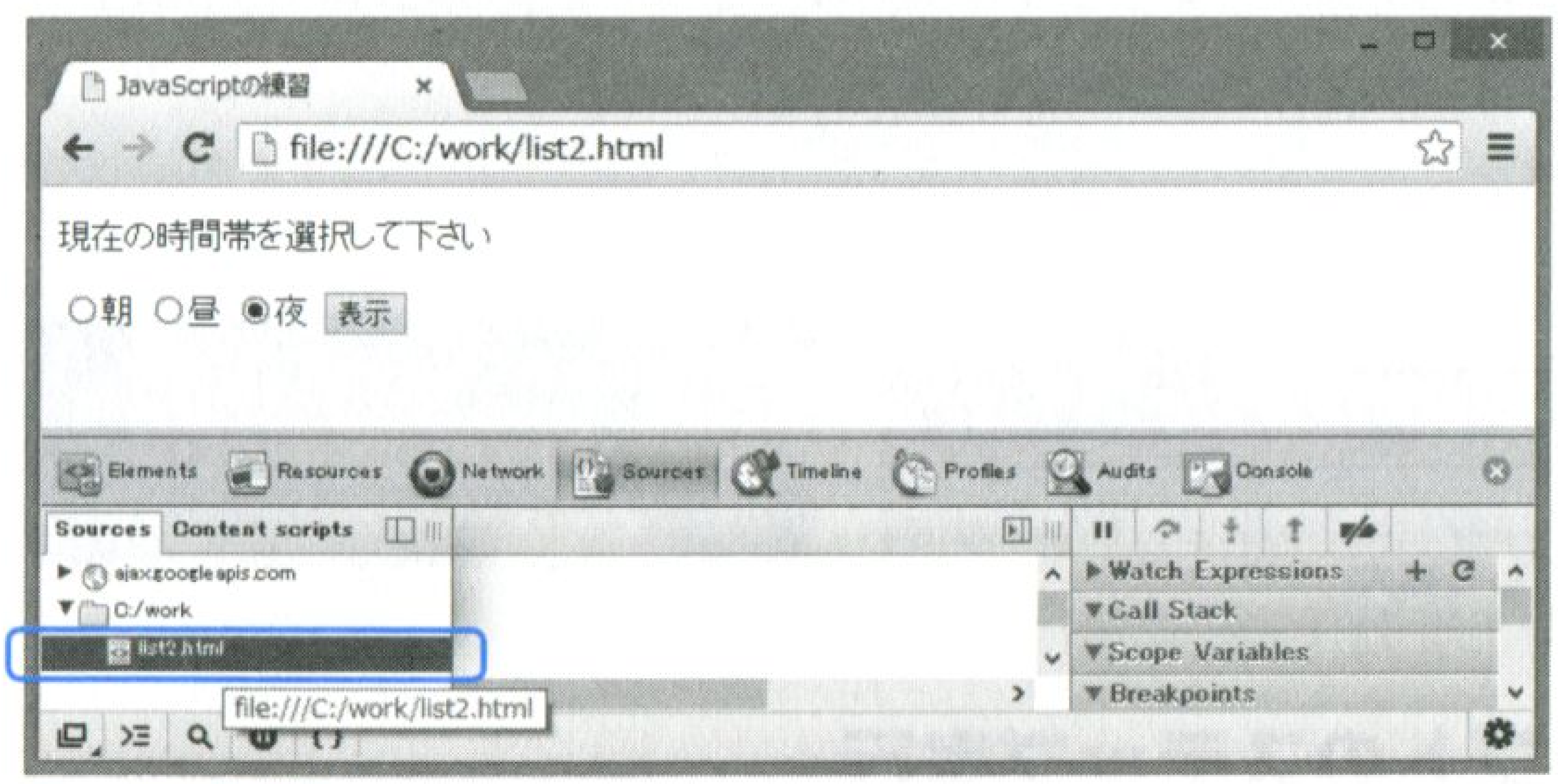
ブレークポイントは、指定したプログラム行でスクリプトの実行を一時停止する機能です。今回は「こんにちは」というメッセージが表示されてしまうことから、showMessage関数が問題と思われる箇所です。そこで、この関数の最初の行にブレークポイントを設定します。

ブレークポイントを設定する手順を見ていきましょう。

❶ JavaScriptコンソールで「Sources」タブを選択し、左上のアイコン［▶］をクリックします。



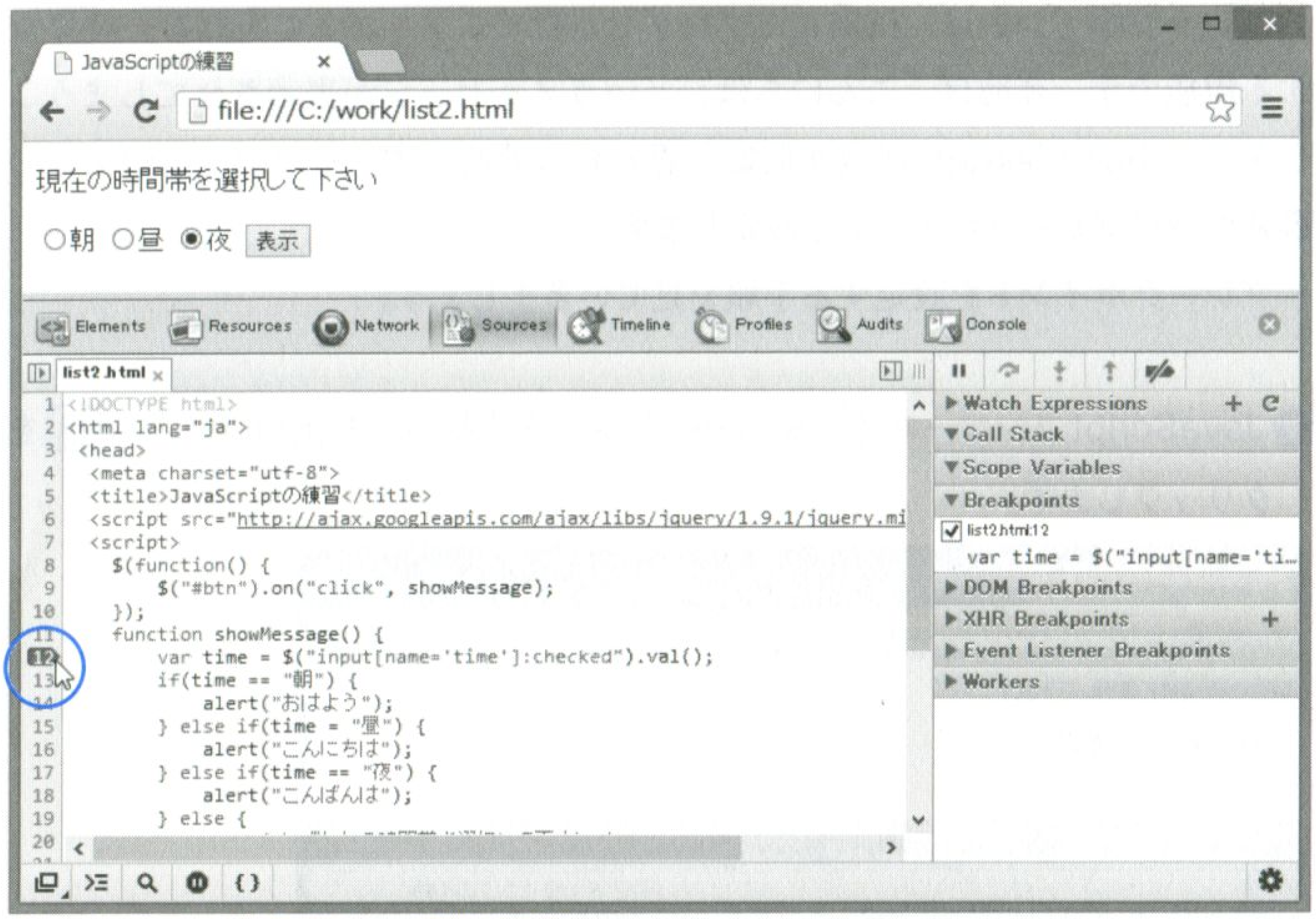
❷ 表示された一覧からスクリプトが記述されているファイル（ここではlist2.html）を選択します。



ファイルを選択

付録

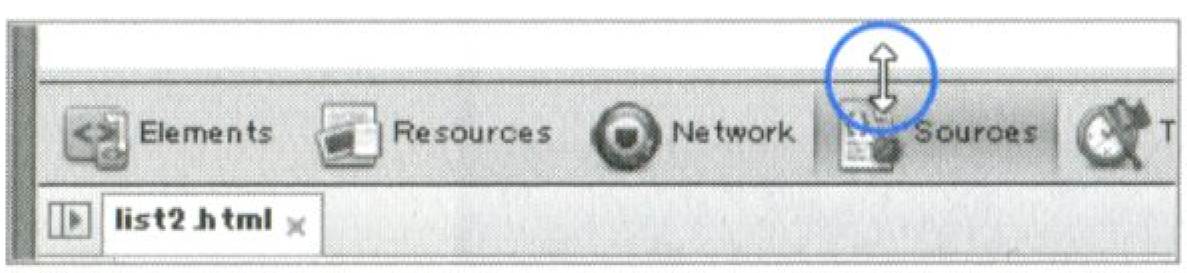
❸ ファイルの内容が表示されたら、ブレークポイントを設定する行の行番号（ここでは12）をクリックします。



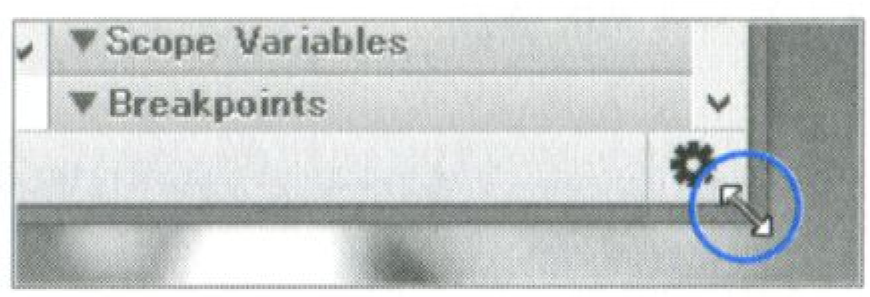
ブレークポイントを設定したい行数を選択

これで12行目にブレークポイントが設定されました。

> **Note** JavaScriptコンソールのウィンドウは、通常のウィンドウと同じく、ウィンドウの左下や上部をドラッグすることで大きくしたり小さくしたりできます。適宜、見やすい大きさに変更して利用してください。



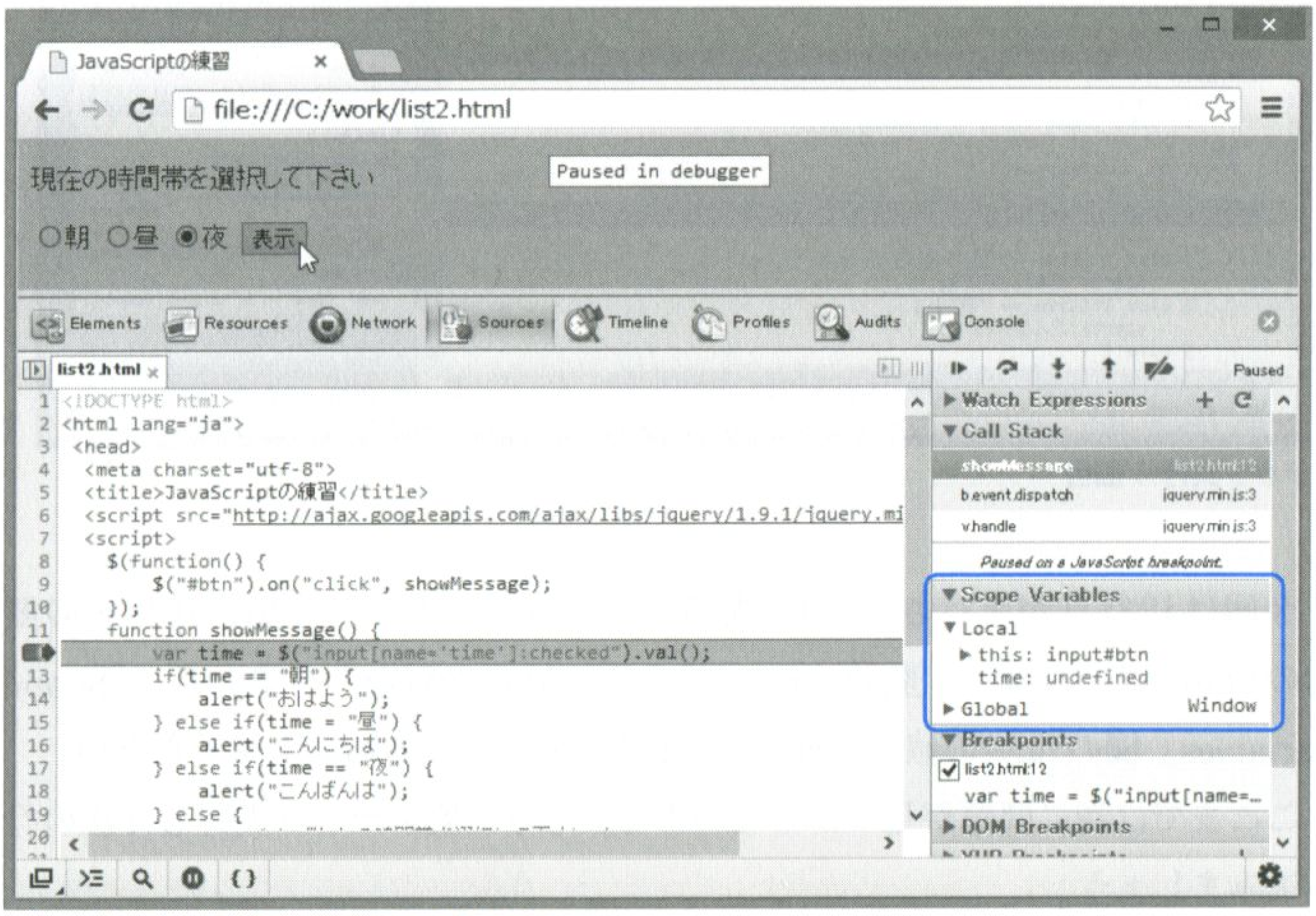
ウィンドウをドラッグして拡大



スクリプトのステップ実行

それでは、オプションボタンで「夜」を選択して［表示］ボタンをクリックし、もう一度スクリプトを実行してみましょう。

すると、次のように、ブレークポイントの設定行でスクリプトが停止します。現在の停止位置は青く表示されます。



ブレークポイント設定行で処理が止まる

このとき、右側の「Scope Variables」には変数の中身が表示されます。12行はまだ実行されていないので、変数timeの値は何も代入されていないことを表わす「undefined」値になっています。

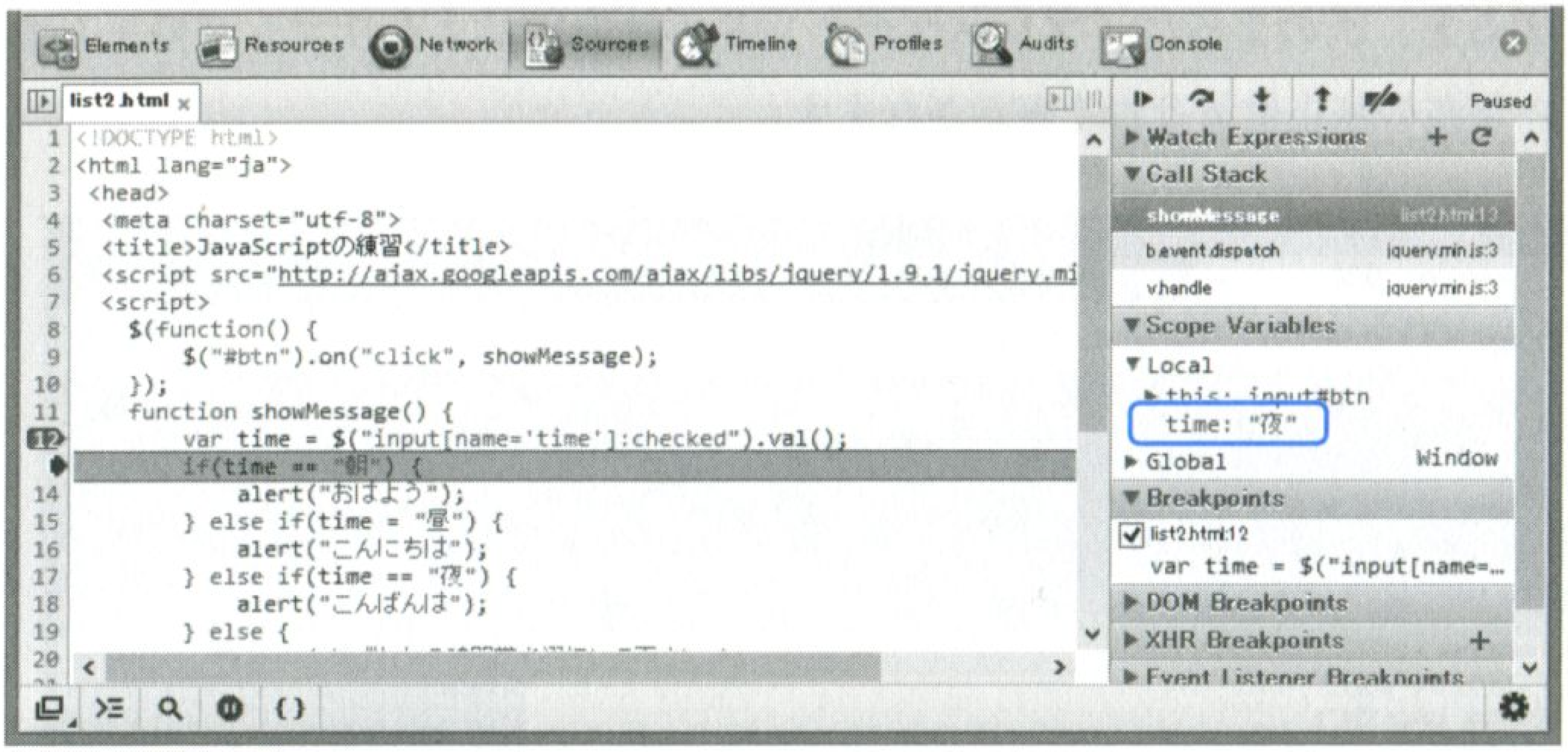
次に**ステップ実行**を行ないましょう。ステップ実行は、スクリプトを1行ずつ実行していく機能です。コンソールの右側にあるアイコン をクリックして実行できます。1回クリックしてください。



デバッグ関連のアイコン

ステップ実行を行なうと、現在の行を実行し、次の行へ進みます。

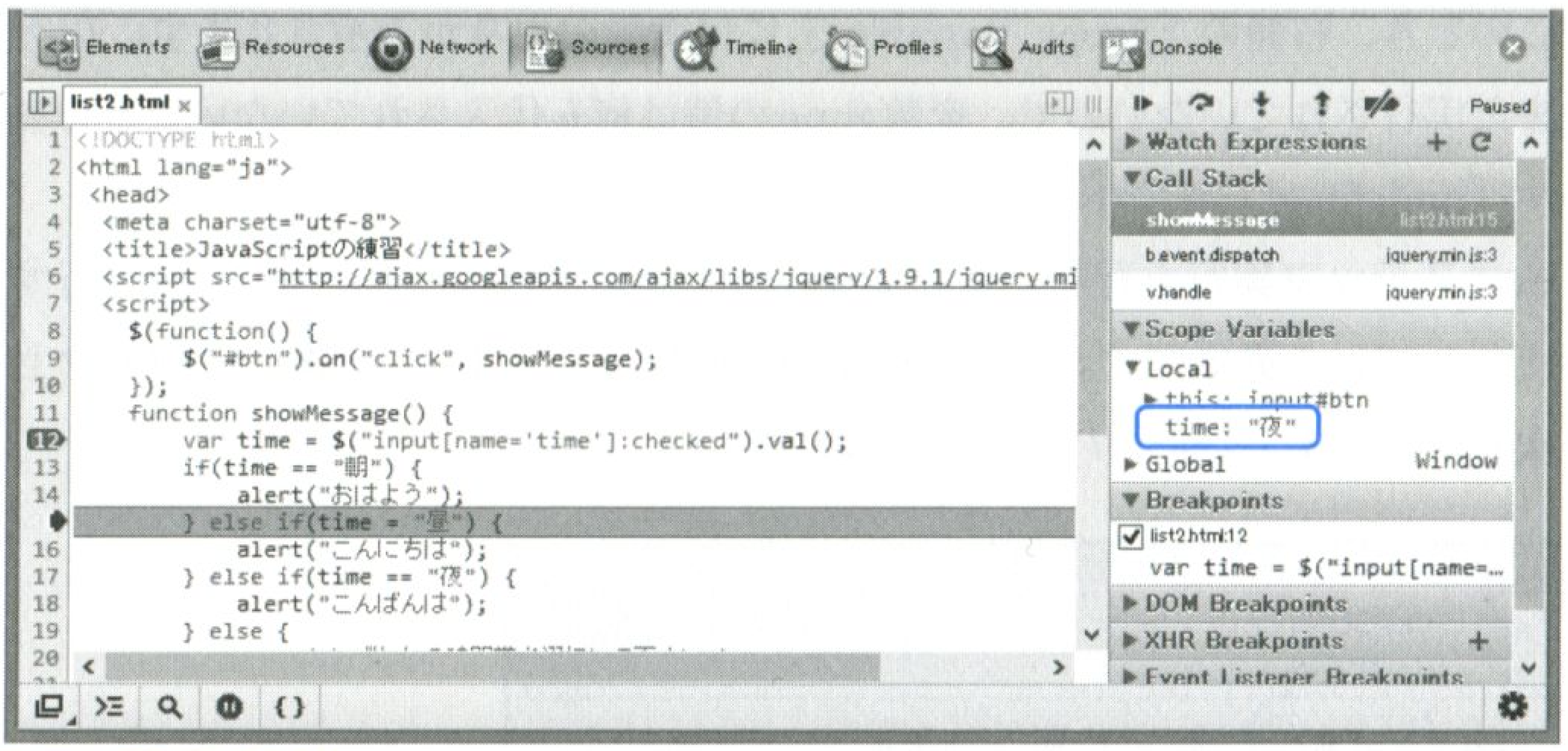
付録



ステップ実行１回目

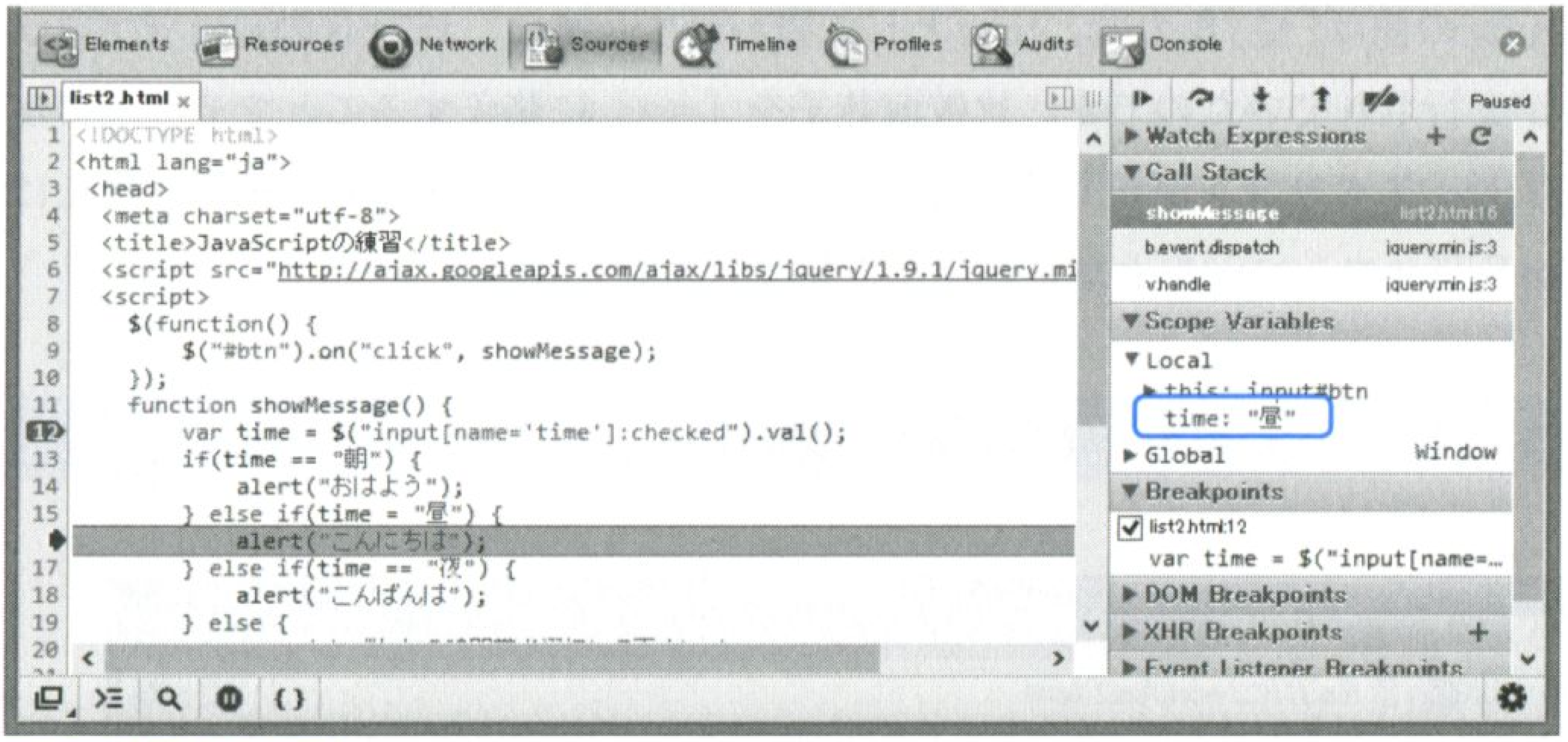
今回は12行目が実行され、3つあるオプションボタンのうち、選択中の要素のvalue値を取得し、変数timeへ代入する処理が行なわれました。「Scope Variables」欄のtimeの値が「夜」に変化していることを確認してください。

ここまでは特に問題が見つからないので、もう一度ステップ実行して次の行へ進みましょう



ステップ実行２回目

timeの値は「朝」ではないので、if文の最初の条件式をスルーして次の条件式へ進みました。さらにステップ実行してみます。



ステップ実行３回目

15行目が実行された結果、timeの値が「昼」に変化してしまいました。ステップ実行しながら変数の値を確認することによって、15行目のelse if文の条件式にバグの原因があることを特定できましたね。

15行目の条件式をよく見てみると、2つの値を比較する演算子が「==」ではなく代入を表わす「=」になってしまっています。

```
15:            } else if(time = "昼") {
```

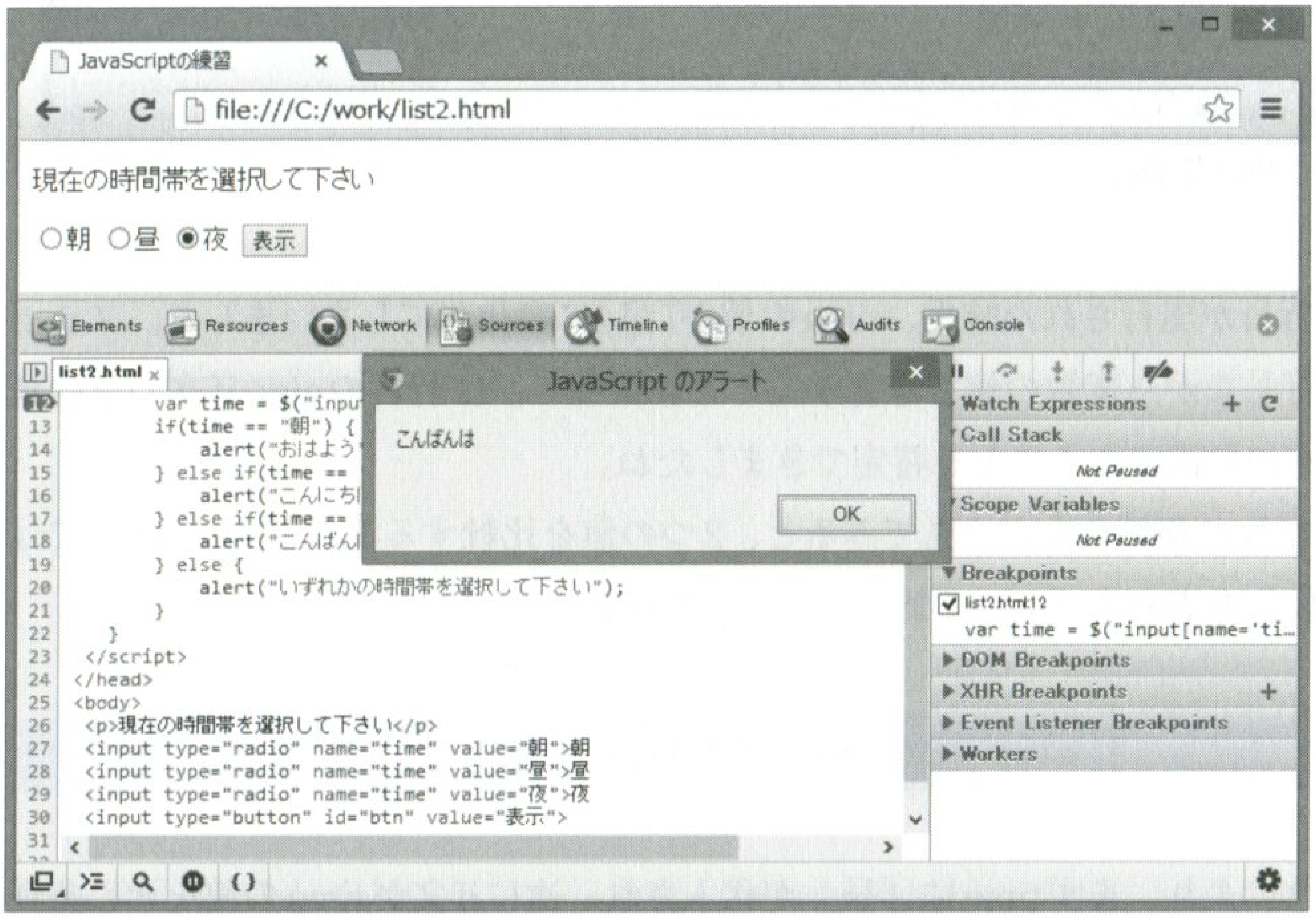
これにより、まずtimeに「昼」が代入され、次にif文がtimeの値を元に条件判定を行ないます。

JavaScriptでは条件式に論理値以外の値が指定された場合、次のように判定されます。

falseとして判定される値	trueとして判定される値
0	左記以外の値
null	
undefined	
NaN	
空文字	

timeの値は「昼」なので、trueとして扱われてしまい、15行目のelse ifブロックが実行されていたのです。比較演算子を「==」に修正することでデバッグが完了し、スクリプトが正しく動作するようになります（ソースコードの修正はJavaScriptコンソール上から行なうことはできません）。

```
15:        } else if(time == "昼") {
```



15行目を変更後の実行結果

このように、原因不明のバグが発生した場合は、ただやみくもにソースコードを目視確認していくよりも、デバッグツールを効率的に活用することが原因特定への近道です。ぜひツールを活用してください。

Column

ステップ実行以外の各種アイコンの機能

ステップ実行以外の機能についても紹介しておきます。実際に試してみると良いでしょう。



❶実行

一時停止中のスクリプトを再開します。ステップ実行を途中で終了し、スクリプトを最後まで実行する場合はこのアイコンをクリックします。

❷ステップイン

ステップ実行と同様に次の処理を実行する機能ですが、関数を呼び出している場合は呼び出し先の関数の中も1行ずつ実行します。

❸ステップアウト

ステップ実行と同様に次の処理を実行する機能ですが、現在実行中の関数を最後まで一気に実行し、呼び出し元へ戻ります。

❹ブレークポイントの無効化

すべてのブレークポイントを一時的に無効化します。もう一度クリックするとブレークポイントが有効になります（ブレークポイントを完全に除去したい場合は、ブレークポイントを設定した行番号をもう一度クリックします）。

■ チェックテストの解答例

第1章

CheckTest1

Q1

A）HTML（Hyper Text Markup Language）

B）CSS（Cascading Style Sheets）

C）JavaScript

Q2

A）パソコン

B）命令

C）簡易的

D）簡易プログラミング言語

Q3

静的なWebページとは、主にHTML（とCSS）で作られた「動きのない」Webページのことです。簡単にいうと、同じURLにアクセスすると、毎回、同じ内容、デザインで表示されるページです。

対して、動的なWebページとは、主にHTML/CSS/JavaScriptで作られた「動きのある」Webページのことです。こちらは、同じURLでも、異なる内容、デザインで表示されるページです。

CheckTest2

Q1

A）リクエスト（要求）

B）ブラウザ

C）レスポンス（応答）

D）Webサーバー

Q2

サーバーサイドスクリプトとは、データベースとの連携を主目的とした、サーバー上で動作するスクリプトのこと。主にPHPやRubyなどのプログラミング言語で記述されます。

クライアントサイドスクリプトとは、Webページの操作を主目的とした、クライアント（ブラウザ）上で動作するスクリプトのこと。主にJavaScriptで記述されます。

この章の到達度チェック

Q1

主にブラウザ上で動作するスクリプト言語（ブラウザに作業を命令するためのプログラミング言語）。

Q2

ブラウザ上

解説 ブラウザ上で動作し、主にWebページの表示を制御します。

Q3

A）クライアント側（またはブラウザ上）

B）サーバー

C）速い

D）ブラウザ

Q4

ブラウザの種類によってスクリプトの動作に違いがあります。

解説 現時点ではこのような問題があるということだけ知識としておぼえておいてください。

Q5

テキストエディタとブラウザ

解説 テキストエディタでスクリプトを記述し、ブラウザで動作確認をします。

第2章

CheckTest1

Q1

A）Webページ

B）タグ

CheckTest2

Q1

A）<

B）>

C）要素

Q2

要素の中に別の要素を入れ込んでいく構造のこと。入れ子とも呼びます。

CheckTest3

Q1

<html>タグ

Q2

charset属性

CheckTest4

Q1

<a>タグ

Q2

src属性

この章の到達度チェック

Q1

utf-8

解説 <meta>タグで指定した文字コードとファイルを保存するときに指定する文字コードを合わせます。

Q2

```
type="radio"
```

解説 type属性の値を変えることで、さまざまな部品を生成できます。

Q3

A）tr

B）th

C）td

解説 行（tr）の中に列（th、td）が入っています。

Q4

```
diary/B.html
```

解説 相対パス指定では、現在のページから見た目的地までの経路を指定します。ディレクトリ間はスラッシュ（/）で区切ります。

Q5

```
../hobby/B.html
```

解説 1つ上のディレクトリへたどるには「..」のようにドット（.）を2つつなげます。この問題ではいったんrootディレクトリまで戻ってから、hobbyディレクトリへ進んでいます。ちなみに、2つ上のディレクトリにあるファイルを参照するには"../../***.html"のように記述します。

第3章

CheckTest1

Q1

正式名称はCascading Style Sheetsで、HTML文書を装飾する（飾り付ける）ための言語。背景や文字の色設定、文字や画像のサイズや表示位置の調整など、Webページにスタイルを適用するために使われます。

Q2

A）link

B）"stylesheet"

C）"text/css"

D）href

CheckTest2

Q1

A）セレクタ
B）プロパティ

Q2

文字サイズを指定するプロパティ。

この章の到達度チェック

Q1

```
<link rel="stylesheet" type="text/css" href="style.css">
```

解説 <link>タグは空要素です。

Q2

```
p {
    color: blue;
}
```

解説 特定のタグすべてに対してスタイルを適用する場合は基本セレクタを使います。

Q3

```
#ex {
    font-size: 15px;
}
```

解説 IDセレクタはID属性値の前に#を付けます。

Q4

```
* {
    margin: 0px;
    padding: 0px;
}
```

解説 波かっこ｛｝内には複数のプロパティを並べて指定できます。

Q5

```
.box {
    width: 200px;
    height: 80px;
}
```

解説 クラスセレクタはクラス名の前にドット（.）を付けます。

第4章

CheckTest1

Q1

<script>タグ

解説 <script>タグは<head>または<body>のどちらの中に記述してもかまいません。

Q2

```
alert("こんにちは");
```

解説 文字列はダブルクォートまたはシングルクォートで囲みます。

CheckTest2

Q1

```
var
```

解説 変数は利用する前にまず宣言をします。

Q2

```
console.log(x);
```

解説 変数名を指定すると、変数の中身が出力されます。

CheckTest3

Q1

100100

解説 ＋演算子で2つの値を加算した場合、どちらか片方でも文字列であれば、値の連結が行なわれます。

Q2

31

解説 yの初期値10に3を掛けて、最後に1加算しています。

CheckTest4

Q1

```
パイナップル
```

解説 インデックスは0から始まります。

Q2

```
5
```

解説 配列名.lengthで配列の要素数が取得できます。

この章の到達度チェック

Q1

```
<script src="sample.js"></script>
```

解説 <script>タグには終了タグが必要です。

Q2

```
20
```

解説 先にyの値をxに代入しているので、xの値は20に上書きされています。

Q3

```
1010
```

解説 数値の10に文字列の"10"を連結した結果、xの値は文字列型の"1010"になります。

Q4

[1][2]

解説 二次元配列の場合、次のようにインデックスを指定します。

```
配列名[ 外側の配列のインデックス ][ 内側の配列のインデックス ]
```

Q5

キー

解説 連想配列はキーと値のセットが複数集まったものです。

Q6

name

解説 連想配列の要素の値を取得するには、角かっこ[]かドット(.)を使います。

Q7

初期値を角かっこ[]で囲んでいるのが誤り。

解説 連想配列の初期値は波かっこ{}で囲んで指定します。

第5章

CheckTest1

Q1

```
NG
```

解説 条件式の内容は「xに代入されている文字列が"ABC"ではない」です。xに入っているのは"ABC"なので、結果はfalseとなります。

Q2

```
NG
```

解説 条件式の内容は「yが10以上でかつ、xがyよりも大きい」です。yが10以上という条件は満たしていますが、xはyよりも小さいため結果はfalseとなります。

CheckTest2

Q1

```
C
D
E
```

解説 swtich文では、break文がない限り、値が一致したcaseキーワード以降の処理をすべて実行します。

Q2

```
E
```

解説 どの値にも一致しなかった場合はdefaultキーワード以降の処理が実行されます。if文のelseキーワードと同様の役割です。

CheckTest3

Q1

3回

解説 aの値が1、2、3と変化していきます。aが3になると、次の繰り返し処理へ進んだタイミングで「aが3未満」という条件を満たさなくなり、繰り返しを抜けます。

Q2

0回

解説 最初からaの値が3なので、「aが3未満」という条件を満たすことがなく、ブロック内は実行されません。

CheckTest4

Q1

```
5
4
3
2
1
```

解説 iが5から始まり、0よりも大きい間繰り返します。繰り返すたびにiの値が1減算されます。

Q2

4回

解説 for-in文は連想配列の要素数分繰り返しを行ないます。

この章の到達度チェック

Q1

```
NG
```

解説 イコールが3つつながると、データ型まで含めて一致するかどうかチェックします。数値型の10と文字列型の10を比較しているので、else以降のブロックが実行されます。

Q2

default

解説 caseで指定されたどの値にも一致しなかった場合に実行されます。

Q3

A）0

B）fruits.length

解説 インデックスは0から始まるので、カウンタ変数の初期値も0にします。また、インデックスの最大値は配列の要素数-1となるので、カウンタ変数の値が配列の要素数未満の間繰り返すようにします。

Q4

A）in

B）[x]

解説 連想配列の要素を1つずつ処理するには、for-in文を使います。変数xには連想配列が持つキーのうちいずれか1つが代入されます。

Q5

```
1
2
4
5
```

解説 iが5よりも大きくなると繰り返しを終了します。また、iが3で割り切れるときは次の繰り返し処理へ進みます。

Q6

```
1
```

解説 iが1から10までの間繰り返すfor文の中で、iが2で割り切れたときに繰り返しから抜けるようになっています。最初にiが2で割り切れるのは2回目なので、それ以降の繰り返し処理は実行されません。

Q7

```
1
3
5
7
9
```

解説 iが1から10までの間繰り返すfor文の中で、iが2で割り切れたときに次の繰り返し処理へ進むようになっています。continue文が実行されると実行中の処理を中断して次の繰り返し処理へ進むので、コンソールへの出力は行なわれません。

第6章

CheckTest1

Q1

```
func("ABC");
```

解説 ()内には引数として渡す値を記述します。数値や文字列、変数や配列などでもかまいません。

Q2

```
return x;
```

解説 戻り値の指定にはreturn文を使います。

CheckTest2

Q1

エンコード

解説 エンコードされた文字列を元に戻すことはデコードと呼びます。

Q2

NaN

解説 数値でないことを表わす特殊な値です。isNaN関数でNaN値かどうかを調べることができます。

CheckTest3

Q1

x) グローバル変数

y) ローカル変数

解説 関数の外側で宣言した変数はグローバル変数、関数の内側で宣言した変数はローカル変数です。

Q2

x) ローカル変数

y) グローバル変数

解説 関数の引数はローカル変数です。また、宣言を省略した変数はグローバル変数になります。

この章の到達度チェック

Q1

calcTriangle(30, 50)

解説 引数が複数ある場合はカンマ区切りで指定します。

Q2

```
OK
```

解説 文字列型の"10.5"を数値型の10.5に変換しています。parseFloat関数を使うと小数点以下の値も得ることができます。isNaN関数は指定された値がNaNのときにtrueを返す関数ですので、この問題では戻り値がfalseとなります。!演算子によって論理値が反転し、if文の条件式の結果はtrueとなります。

Q3

```
30
```

解説 変数xはグローバル変数です。plus関数内でxに20を加算しているので、値が30に上書きされます。

Q4

```
10
```

解説 plus関数の引数xはローカル変数で、グローバル変数のxとは名前が同じだけの別物です。plus関数内での加算処理はローカル変数xに対して行なっているものです。よってグローバル変数xの値は変化しません。

Q5

func

解説 定義した関数を変数へ代入するには、関数名だけを指定します。

Q6

```
f("ABC");
```

解説 変数fに匿名関数が代入されています。

Q7

("こんにちは")

解説 定義した関数をすぐに呼び出して実行する、即時関数のパターンです。後ろの丸かっこ内に指定した値が引数として渡されます。

第7章

CheckTest1

Q1

```
obj.x();
```

解説 メソッドを実行するには、キーの後ろに丸かっこ()を付けます。

Q2

this

解説 自身のオブジェクトが所有しているプロパティやメソッドを利用する場合に先頭に付けます。

CheckTest2

Q1

Array

解説 [ ]で初期化するとArrayオブジェクトになります。

Q2

Object

解説 { }で初期化するとObjectオブジェクトになります。

この章の到達度チェック

Q1

A）item2

B）item1

解説 メソッドの所有者を先に記述します。

Q2

```
50
```

解説 callメソッドによって別のオブジェクトのメソッドを実行した場合、thisキーワードが表わすものはcallメソッドの引数として渡されたオブジェクトです。

Q3

split

解説 変数strには、連結された文字列"A/B/C"が代入されています。splitメソッドで"/"を区切り文字として分解すれば、元の配列と同じ状態に戻ります。

Q4

```
4
```

解説 "mon"という文字列と一致する箇所は4か所あるので、matchメソッドは要素4つ分の配列を返します。

Q5

```
friday saturday sunflower
```

解説 正規表現のパターンに$が付いているので、末尾の"day"のみが置換対象となります。

第8章

CheckTest1

Q1

A）イベントハンドラ

B）イベント駆動型モデル

CheckTest2

Q1

<body>要素

Q2

blurイベント

この章の到達度チェック

Q1

onload

解説 onloadは、HTMLタグの読み込みがすべて終わった後で実行されます。

Q2

onblur

解説 onblurは、対象の部品の選択状態が解除されたタイミングで実行されます。

Q3

onchange

解説 onchangeは、セレクトボックス等の選択項目が変更されたタイミングで実行されます。

Q4

this

解説 onclickなどのイベントハンドラの属性値として記述した処理の中では、thisはHTML要素自身を表わします。

Q5

value

解説 要素の属性値を得る場合は、要素が代入されている変数名の後ろにドットを挟んで属性名をつなげます。

第9章

CheckTest1

Q1

A）親

B）ウィンドウ全体

CheckTest2

Q1

```
window.close();
```

Q2

setTimeoutメソッド

CheckTest3

Q1

```
alert(location.href);
```

Q2

```
history.back();
```

この章の到達度チェック

Q1

前のページへ戻ります。

解説 confirmメソッドは、OKがクリックされた場合に戻り値としてtrueを返します。この問題では!演算子が付いているので、論理値が反転してfalseとなります。よってelseブロックが実行されます。

Q2

http://www.google.co.jp/ に移動します。

解説 Locationオブジェクトのhrefプロパティは現在のURLを表わします。これを書き換えることで、別のページに移動することができます。

Q3

5秒おき

解説 タイマー処理の実行間隔は、ミリ秒単位で記述します。

Q4

```
clearInterval(tid);
```

解説 clearIntervalメソッドは、引数に停止するタイマーのIDを指定します。

Q5

```
document.fr.tx.value = "ABC";
```

解説 「document.フォーム名.フォーム内部品名」の順番に指定することで要素を取得できます。要素が持つ属性値はスクリプト上で変更できます。

第10章

CheckTest1

Q1

A) マークアップ言語

B) ツリー

Q2

3. 兄弟要素

CheckTest2

Q1

HTML要素の集合体で、配列のように0から始まるインデックスが振られています。

Q2

```
alert(document.getElementById("message").innerHTML);
```

この章の到達度チェック

Q1

getElementById

解説 idを指定して1件の要素を取得します。

Q2

A) getElementsByName

B) [0]

解説 getElementsByNameは、対象の要素が１件の場合でも配列と同様の形式で戻り値を返します。よってインデックスを指定する必要があります。

Q3

className

解説 class属性を変更するには、classNameプロパティを利用します。

Q4

+=

解説 要素が持っている内容の後ろにHTML文字列を挿入する場合は、+=演算子でinnerHTMLに連結させます。=演算子を使うと内容を置き換えてしまうので注意しましょう。

Q5

```
window.onload = func;
```

解説 onloadイベントハンドラのみ、window.onloadと記述します。定義済みの関数を設定するには関数名だけを記述して、丸かっこ（ ）はつけません。

第11章

CheckTest1

Q1

クロスブラウザ問題

解説 jQueryは、ブラウザの違いを吸収し、クロスブラウザ問題を解決してくれます。

CheckTest2

Q1

<script>要素でjQuery（jquery-x.x.x.min.js）を読み込みます。

この章の到達度チェック

Q1

LightBox系プラグイン

解説 jQueryプラグインの中でも、LightBox系プラグインは非常に多くの種類が開発されています。

Q2

```
<script src="scripts/jquery-1.9.1.min.js"></script>
```

解説 終了タグが必要です。

第12章

CheckTest1

Q1

ページ上のHTMLタグが解析され、DOMが利用可能となったタイミング。

Q2

```
$(function(){ });
```

CheckTest2

Q1

div p

解説 「<div>要素の中のすべての<p>要素」が対象なので、孫要素以下も取得できる子孫セレクタを使います。

Q2

div > *

解説 すべての子要素に対して処理を実行するには、子セレクタの子要素を全称セレクタで指定します。

CheckTest3

Q1

:first

解説 div要素の子要素の中で、最も先頭の要素を抽出します。

Q2

[name='hoge']

解説 id、class以外の属性を条件として要素を検索するには、属性フィルタを利用します。

CheckTest4

Q1

A) on

B) html

解説 onメソッドでイベントハンドラを登録し、htmlメソッドで要素の内容を操作します。

Q2

A) attr

B) name

解説 value以外の属性値を操作するには、attrメソッドを使います。

この章の到達度チェック

Q1

p,#hoge

解説 複数の条件で要素を検索する場合は、グループセレクタを使います。

Q2

off

解説 offメソッドは、引数を省略した場合すべてのイベントハンドラを削除します。

Q3

.filter

解説 要素群に対して絞り込みをかけるにはfilterメソッドを利用します。メソッドチェーンを利用すると、セレクタで取得した要素に対していくつもの処理を実行することができます。

Q4

A） click

B） toggleClass

解説 toggleClassメソッドは、呼び出されるたびにCSSクラスの追加と削除を交互に実行します。

Q5

replaceWith

解説 ある要素を異なる要素に置き換える場合は、replaceWithメソッドを使います。

Q6

append

解説 要素が持つ子要素の末尾に項目を追加します。先頭に追加する場合はprependメソッドを使います。

Q7

A） empty

B） after

解説 emptyメソッドは子要素をすべて削除し、afterメソッドは要素の後ろに兄弟要素を追加します。

第13章

CheckTest1

Q1

load

解説 loadメソッドはHTMLを読み込み、要素に挿入します。

Q2

```
$("div").load("sample.html #hoge");
```

解説 URLの後ろにスペースを挟んでセレクタを記述します。

この章の到達度チェック

Q1

JSON

解説 JSONデータは配列やオブジェクトとして扱うことができます。

Q2

JSONP

解説 外部ドメインからJSONは取得できないので、JSONPを利用します。

Q3

$.getJSON

解説 $. から始まります。

Q4

プロパティ名がダブルクォートで囲まれていません。また、シングルクォート（'）が利用されています。

解説 JSONは、文字列はすべてダブルクォート（"）で囲みます。

Q5

クロスドメイン通信

解説 JSONPを使えば、クロスドメイン通信を実現することができます。

Q6

callback=?

解説 ?の部分はgetJSONメソッドの2つ目の引数で指定したコールバック関数に置換されます。

第14章

CheckTest1

Q1

toggle

解説 toggleメソッドは、実行するたびに要素の表示／非表示を切り替えます。

Q2

1000

解説 アニメーションの速さはミリ秒単位で指定します。その他に"slow"、"normal"、"fast"があります。

この章の到達度チェック

Q1

"width" : "400px", "height" : "250px"

解説 CSSプロパティ間はカンマで区切ります。

Q2

「一定速度で動く」という意味です。

解説 イージングにはlinearとswingがあります。

Q3

300px

解説 animateメソッドでは、複合代入演算子を使ってCSSプロパティ値に加減算を行なうことができます。

Q4

animate

解説 animateメソッドをメソッドチェーンでつなげて実行すると、複数のアニメーションを順番に実行することができます。

Q5

なにも表示されません。

解説 ページが表示されると同時にhideメソッドで<img>要素を非表示にしているので、その後のanimateメソッドの動きは見ることができません。

Q6

<img>要素が徐々に大きくなり、幅が400pxになると見えなくなります。

解説 コールバック関数のtoPassiveの中で<img>要素を非表示にしています。コールバック関数はアニメーションが完了したタイミングで実行されます。

索引

タグ

<!-- ～ --> 36
<!DOCTYPE html> 33
<= 111
<a> 35, 36
<body> 34
　～要素の中に記述する要素 35

 36, 49
<button> 35
<div> 35
<form> 46
<h1> 35
<head> 33
<html lang="ja"> 33
<html> 33
<img> 36, 40
<input> 47
 42
<link> 56
<meta charset="utf8"> 34
<meta> 34
 42
<option> 50
<p> 35
<script> 71, 73
<select> 50
<table> 44
<td> 44
<th> 44
<title> 34
<tr> 44
 42

記号

' 28, 77, 88
- 90
-- 91, 92
! 113, 114
!= 111
!== 111
"" 28, 77, 88
62, 271
% 90, 91
%= 91
&& 113
( ) 87
* 66, 90, 271
*= 91
.css 56
.html 23
/ 90
/* ～ */ 61, 76
// 76
/= 91
:checked 279
:contains 279, 282
:eq 279, 283
:even 279, 281
:first 279, 280
:last 279, 280
:not 279, 282
:odd 279, 281
:selected 279
; 72
[ ] 95, 100, 176
{ } 72, 102
|| 113, 114
+ 90, 91, 92
++ 91, 92
+= 91, 92
< 111
= 86, 112
-= 91
== 111, 112
=== 111
> 111
>= 111

A

Ajax 319
Arrayオブジェクト 176
action属性 46
addClassメソッド　jQuery 302

afterメソッド jQuery 308
alertメソッド 77, 210
alt属性 36
animateメソッド jQuery 345
　～に対するメソッドチェーン 347
appendメソッド jQuery 307, 312
applyメソッド 171
attrメソッド jQuery 293

B

backgroundｰcolorプロパティ 60
backメソッド 218
beforeメソッド jQuery 308
blurイベント 192, 198
break文 119, 132

C

callメソッド 169
ceilメソッド 175
changeイベント 198
checked属性 47
childNodesプロパティ 254
classNameプロパティ 234
class属性 36, 64
clearIntervalメソッド 215
clickイベント 193
closeメソッド 212, 213
colorプロパティ 58, 59, 60
confirmオブジェクト 210
console.log 77
continue文 133
CSS 6, 55
　～クラスの変更 235
　～クラスの変更 jQuery 302
　～ファイルの読み込み 56
　記述時の注意点 59
　記述する場所 56
　クラス 63
　セレクタ 58, 61
　セレクタ jQuery 271
　プロパティ 58, 60

D

Dateオブジェクト 174
decodeURIComponent 149, 152
disabled属性 47, 355
displayプロパティ 60
Documentオブジェクト 221
　要素の内容の操作 237
　要素へのアクセス 231
DOM 229

E

else 109
else if 115
emptyメソッド jQuery 309
enabled属性 47
encodeURIComponent関数 149, 151

F

fadeInメソッド jQuery 342
fadeOutメソッド jQuery 342
false 89, 110
filterメソッド jQuery 286
findメソッド jQuery 288
floorメソッド 175
focusイベント 192, 198
font-sizeプロパティ 58, 59, 60
for-in文 128
for文 125
　～によるカウントダウン 127
function文 142, 144, 145

G

getDateメソッド 174
getElementByIdメソッド 231, 232
getElementsByNameメソッド 231, 233
getFullYearメソッド 174
getHoursメソッド 174
getJSONメソッド jQuery 326, 331
getMinutesメソッド 174
getMonthメソッド 174

getSecondsメソッド 174

H

heightプロパティ 60
height属性 36
hideメソッド jQuery 337, 338
history.back() 218
Historyオブジェクト 218
href属性 35, 36
hrefプロパティ 218
HTML 6, 23
　～の表示 32
　～の入力 30
　～の保存 31
　記述時の注意点 26
　基本的な書き方 26
　部品 42
HTMLタグ →タグ
HTMLファイル 23
　～名の注意点 32
HTML文書 23
htmlメソッド jQuery 291

I

IDセレクタ 62
　jQuery 271, 272
　クラスセレクタとの違い 65
id（ID）属性 36, 62
if文 109
indexOfメソッド 178
innerHTMLプロパティ 237
isNaN 149, 150

J

JavaScript 3, 6
　～ファイルの読み込み 74
　開発環境 14
　書き方のルール 72
　組み込みオブジェクト 173
　組み込み関数 149
　データ型 89
　記述する場所 73
JavaScriptコンソール 78, 363, 364
　ステップ実行 369
　デバッグ機能 373
　ブレークポイント 367
joinメソッド 177
jQuery 254
　～プラグイン 261
　Ajax 319
　アニメーション 337
　イベント発生元の特定 300
　イベントハンドラの設定 295
　基本の書き方 269
　セレクタ 271
　内容と属性の操作 291
　フィルタ 279
　要素の絞り込み 284
　要素の取得 271
　要素の挿入 307
　要素の抽出 279
　読み込み 257
　ライセンス 255
JSON 325
　～ファイル 326
JSONP 329, 330

L

leftプロパティ 60, 351
lengthプロパティ 99, 176
LightBox系プラグイン 261
loadイベント 195
loadメソッド jQuery 321
location.href 218
Locationオブジェクト 218

M

marginプロパティ 60
matchメソッド 180, 186
Mathオブジェクト 175
method属性 46
mouseoutイベント 197
mouseoverイベント 197

N

name属性 …… 46, 47, 221
NaN …… 89, 150
new …… 174
null …… 89, 243

O

Objectオブジェクト …… 173
offメソッド jQuery …… 299
onblur …… 192, 198
onchange …… 192, 199
onclick …… 192, 193, 194
onfocus …… 192, 198
onload …… 192, 196
　～イベントハンドラの設定 …… 244
onmouseout …… 192, 197
onmouseover …… 192, 197
onメソッド jQuery …… 296
openメソッド …… 212
overflowプロパティ …… 352

P

paddingプロパティ …… 60
parseFloat …… 149, 150
parseInt …… 149
positionプロパティ …… 351
prependメソッド jQuery …… 307
pushメソッド …… 176

R

readyメソッド jQuery …… 269, 270
rel属性 …… 56
removeAttrメソッド jQuery …… 294
removeClassメソッド jQuery …… 303
replaceメソッド …… 181, 186
replaceWithメソッド jQuery …… 310
return文 …… 146
reverseメソッド …… 177
roundメソッド …… 175

S

selected属性 …… 50
setIntervalメソッド …… 215
showメソッド jQuery …… 337
sliceメソッド …… 179
slideDownメソッド jQuery …… 340, 341
slideUpメソッド jQuery …… 340, 341
sortメソッド …… 178
splitメソッド …… 180
src属性 …… 36, 40
Stringオブジェクト …… 178, 186
switch文 …… 118
SyntaxError …… 363, 364

T

target属性 …… 35
thisキーワード …… 169
　イベント発生元の特定 …… 200, 300
toggleClassメソッド jQuery …… 304
toggleメソッド jQuery …… 337, 338, 340
topプロパティ …… 60
true …… 89, 110
type属性 …… 47

U

undefined …… 87 ,89, 115
URLエンコード …… 149, 151
URLデコード …… 149, 151
UTF-8 …… 31, 32

V

value属性 …… 47, 50
　～値の取得 jQuery …… 293
　～値の変更 jQuery …… 292
valメソッド jQuery …… 292
var …… 84 , 87, 168
　配列の宣言 …… 95

W

Web API …… 330
Webアプリケーション …… 11
Webサーバー …… 10
Webサイト …… 5
Webページ …… 5
　～内の改行 …… 49
　URL 情報 …… 218
　作成するための技術 …… 6
　静的な～ …… 7
　動的な～ …… 7
　表示情報 …… 221
　履歴情報 …… 218
while文 …… 122
white-spaceプロパティ …… 351
widthプロパティ …… 60
width属性 …… 36
Windowオブジェクト …… 208, 210

あ行

アクセス …… 221
値 …… 77
　～の出力 …… 77
アニメーション　jQuery …… 337
　キュー …… 354
　上下にスライドする …… 340
　スライドショーの作成 …… 348
　独自～の作成 …… 345
　左上を基点に切り替える …… 337
　ぼんやり透けていく …… 342

い行

イベント …… 191
イベント発生元の特定 …… 200
　jQuery …… 300
イベント駆動型モデル …… 191
イベントハンドラ …… 191
　～の設定　jQuery …… 295
　～の変更 …… 242
　JavaScriptによる記述方法 …… 241
　引数を持つ関数を設定 …… 246
入れ子 …… 28
インクリメント …… 92
インターネットの仕組み …… 10
インデックス …… 94, 97
インデント …… 31, 73

う行

ウィンドウ全体の情報 …… 210

え行

エフェクト …… 337
エンコード …… 149, 151
演算子 …… 90

お行

オープンソース …… 255
　～ライセンス …… 265
オブジェクト …… 167
　組み込み～ …… 173
オプションボタン …… 47

か行

改行 …… 49
開始タグ …… 27
階層セレクタ　jQuery …… 273
カウンタ変数 …… 125
返り値 …… 145
隠しフィールド …… 47
拡張子 …… 18
　～を表示する設定 …… 18
画像の表示 …… 40
空文字 …… 115
空要素 …… 27
　～のタグ …… 36
関数 …… 141
　～の代入 …… 159
　組み込み～ …… 149
　コールバック～ …… 326
　即時～ …… 161
　匿名～ …… 161

　　引数のある～ 144
　　引数も戻り値もない～ 142
　　戻り値のある～ 145
関数式 160
間接セレクタ　jQuery 273, 275

き行

キー 102
基本セレクタ 61
キュー 354

く行

組み込みオブジェクト 173
組み込み関数 141, 149
クライアント 10
　　～PC 175
クライアントサイドスクリプト 11
　　メリット／デメリット 12
クラス 63
クラスセレクタ 63
　　IDセレクタとの違い 65
　　jQuery 271
繰り返し 122
　　～の制御 132
グループセレクタ 271
グローバルスコープ 153
グローバル変数 153
クロスドメイン 329
クロスブラウザ問題 253

こ行

構文 109
　　～エラー 363
コード 30
コールバック関数 326
子セレクタ　jQuery 273, 274
コメント
　　CSS 61
　　HTML 36
　　JavaScript 76
コンソール 77
　　Chromeでの利用 78
　　Internet Explorerでの利用 80

さ行

サーバー 10
サーバーサイドスクリプト 11
算術演算子 90

し行

自作関数 141
子孫セレクタ　jQuery 273, 274
終了タグ 27
順次 108
条件式 110
初期化 87
　　配列 95
シングルクォート 28, 77, 88

す行

数値 89
　　～変換 149
スクリプト 4, 30
　　クライアントサイド～ 11
　　サーバーサイド～ 11
スクリプト言語 3
スコープ 153
スタイル 55
スタイルシート 55
　　～言語 55
ステップアウト 373
ステップイン 373
ステップ実行 369
スペース 26
スライドショーの作成　jQuery 348

せ行

正規表現 182
　　パターン 183, 184
静的なWebページ 7
絶対パス 37

セレクタ 58, 61, 67
jQuery 271
セレクトボックス 49
宣言 84
配列 95
全称セレクタ 65
jQuery 271

そ行

送信ボタン 47
相対パス 37, 38
添え字 94
ソース 30
ソースコード 30
即時関数 161
属性 27
～値の取得 jQuery 293
～値の変更 234
～値の変更 jQuery 293
～の削除 jQuery 294
イベントハンドラ 193
属性フィルタ jQuery 284

た行

代入 86
関数の～ 159
タイマー 215
タグ 23
主要な～ 35
特殊な～ 42
ダブルクォート 28, 77, 88
多方向分岐 115

ち行

チェックボックス 47

て行

ディレクトリ 38
～名の注意点 41
データ 83
データ型 88
テーブル 44
テキストエディタ 14
テキストファイル 14
テキストボックス 47
デクリメント 92
デコード 149, 151
デバッグ 362

と行

同期通信 319
動的なWebページ 7
匿名関数 161
ドメイン 329

に行

二次元配列 99
入力チェック 182, 221

ね行

ネスト 28

は行

配列 89, 94
宣言と初期化 95
二次元～ 99
要素の参照 96
要素への代入 97
連想～ 102
バグ 362
パス 38
パターン 183, 184
反復 122
汎用ボタン 47

ひ行

比較演算子 110
引数 141, 144
非同期通信 319

ふ行

ファイル名の注意点 …… 41
フィルタ …… 279
　:checked …… 279
　:contains …… 279, 282
　:eq …… 279, 283
　:even …… 279, 281
　:first …… 279, 280
　:last …… 279, 280
　:not …… 279, 282
　:odd …… 279, 281
　:selected …… 279
フォーカス …… 198
フォーム …… 45
　～要素の取得 …… 221
複合代入演算子 …… 91
部品 …… 42
ブラウザ …… 3, 14, 17
ブラウザオブジェクト …… 207
プラグイン …… 261
　LightBox系～ …… 261
ブレークポイント …… 367
プログラミング …… 4
プログラミング言語 …… 3
プログラム …… 3, 30
ブロック …… 72
プロパティ …… 167
　CSS …… 58, 60
分岐 …… 109

へ行

返却値 …… 145
変数 …… 84
　～の初期化 …… 87
　～の命名規則 …… 85
　値の代入 …… 86
　宣言 …… 84
　中身の表示 …… 87

ほ行

ボタン …… 47

ま行

マークアップ言語 …… 23

め行

メソッド …… 167
　作成と呼び出し …… 168
　別のオブジェクトの～を利用 …… 169
メソッドチェーン …… 287
メモ帳 …… 14
　起動方法 …… 15
メモリ …… 83

も行

文字コード …… 31
文字列 …… 23
　データ型 …… 89
戻り値 …… 141, 145

よ行

要素　HTML …… 27
　～の絞り込み　jQuery …… 284
　～の取得　jQuery …… 271
　～の挿入　jQuery …… 307
　～の抽出　jQuery …… 279
　～の内容の操作 …… 237
　～へのアクセス …… 231
　内容と属性の操作　jQuery …… 291
要素　配列 …… 94
　～数の取得 …… 99
　～の追加 …… 97
要素セレクタ　jQuery …… 271
予約語 …… 85

ら行

ライセンス …… 255
　オープンソース～ …… 265
ライブラリ …… 255
ラジオボタン …… 47

り行

リクエスト …… 10
リクエストパラメータ …… 46, 48, 330
リスト …… 42
リセットボタン …… 47
リンク …… 24, 36
隣接セレクタ　jQuery …… 273, 275

る行

ループ …… 122

れ行

レスポンス …… 10
連想配列 …… 89, 102
　宣言と初期化 …… 102
　要素の参照 …… 103
　要素への代入 …… 105

ろ行

ローカルPC …… 257
ローカルスコープ …… 153
ローカル変数 …… 153
論理演算子 …… 113
論理値 …… 89

著者紹介

生形 可奈子（うぶかた かなこ）

システム開発会社にて業務系／制御系SEを経験した後、教育の楽しさに気づきプログラミング講師に転身。現在はアシアル株式会社のスクール事業部や、専門学校東京テクニカルカレッジ等で教鞭を執っている。自身が初めてプログラムに触れたときに覚えた感動を伝えるべく、日々奮励努力している。

●装丁　イイタカデザイン　飯高 勉
●本文デザイン＆DTP　株式会社マップス

スラスラわかるJavaScript（ジャバスクリプト）

2013年3月15日　初版第1刷発行
2017年4月10日　初版第4刷発行

著　者　アシアル株式会社
生形 可奈子（うぶかた かなこ）

発行人　佐々木 幹夫
発行所　株式会社 翔泳社（http://www.shoeisha.co.jp）
印刷・製本　株式会社 ワコープラネット

ISBN978-4-7981-2583-1　Printed in Japan

Beginner's Best Guide to Programming

スラスラわかる JavaScript

この本で学ぶこと

第 1 章 基礎知識と準備
第 2 章 HTML
第 3 章 CSS
第 4 章 基本文法
第 5 章 制御構文
第 6 章 関数
第 7 章 オブジェクトの基礎
第 8 章 イベント
第 9 章 ブラウザオブジェクト
第10章 DOM
第11章 jQueryの基礎知識
第12章 jQueryの文法
第13章 Ajax
第14章 アニメーション
付　録 プログラムがうまく動かないときの対処法～デバッグ

知識ゼロからプログラムが作れる！



9784798125831
1923055023000
ISBN978-4-7981-2583-1
C3055 ¥2300E
株式会社翔泳社
定価：本体2,300円+税

eco Project
地球にやさしい本づくりを目指します

SE SHOEISHA

Beginner's Best Guide to Programming

スラスラわかるJavaScript

知識ゼロからプログラムが作れる！

アシアル株式会社
生形 可奈子 著

SE SHOEISHA